Int. J. Man-Machine Studies (1977) **9**, 537–581

Hendrix's model for simultaneous actions and continuous processes: an introduction and implementation

JOHN D. LOWRANCE† AND DANIEL P. FRIEDMAN
Computer Science Department, Indiana University, Bloomington, Indiana 47401, U.S.A.

(*Received 21 April 1977*)

This paper presents a self-contained introduction and implementation description to a simulation system for modeling simultaneous actions and continuous processes (Hendrix, 1973). The essence of the system is described by a portion of its abstract:

> A new methodology for the construction of world models is presented. The central feature of this methodology is a mechanism which makes possible the modeling of (1) simultaneous, interactive processes, (2) processes characterized by a continuum of gradual change, (3) involuntarily activated processes (such as the growing of grass) and (4) time as a continuous phenomena.

and by a recent review, Gaines (1975):

> This is a fascinating paper that will be of interest outside the "artificial intelligence" (AI) context in which it is written, from those concerned with simulating and controling multi-element systems to those interested in operational definitions of concepts such as "causality."

Three robot world models are incrementally developed, each introducing a new modeling concept. World models, including a robot world (with sample output), electrical world, and a Turing world are also presented. The interactive operating environment presented permits the user to inspect and alter the run-time structure. A detailed account of the implementation is presented.

1. Overview

We have chosen to base our introductory presentation primarily on robot systems since it is a significant problem in the field of Artificial Intelligence and requires both active and passive agents to be modeled. A robot system is an artificially intelligent entity which has some understanding of the world. Such an understanding is made possible through a model that the robot maintains. The model is a representation of the world as it changes through time. At any particular moment, the robot can reference its model and know the presumed state of the world.

Because of the complexities and vastness of the world, one can attempt to model only a portion of it. The portion chosen is as small as possible yet still allows the robot to perform the desired functions. For example, if a robot's work never takes it out of one particular room, it probably need not know any more of the world than that room. In addition, the information about that room need only be specific enough to accomplish the required tasks.

†Current address: Computer and Information Science, University of Massachusetts at Amherst, Amherst, Massachusetts 01003, U.S.A.

The initial task of model construction is to determine the representation of the world in which the robot is placed. This representation is like a photograph taken at the moment the robot is activated. It describes the world's components and the relationships that exist between them. The information associated with a component might include the type of object, its name, its location, its relationship to other objects, and other relevant characteristics. For example, a light might be described with a switch named SW1, at location LOC that controls light LT1 and is presently in the DOWN position. Each characteristic is a separate entry in a list which forms the initial world model.

If the initial world model is properly constructed, the instant the robot is activated it "knows" about its world. As the world around it changes, the model must be altered accordingly. This is usually done by applying the appropriate operator from an operator list. Each operator models some process which changes the world. The operators describe how the components and their relationships change. In order to keep the model world consistent the appropriate operators need to be available and applied at the correct times.

The nature of these operators in earlier robot systems limits their applicability (Fikes & Nilsson, 1971; Siklóssy & Dreussi, 1973; Siklóssy & Roach, 1973; Winograd, 1972). A number of them share in the same limiting factors. One such factor occurs when only those processes in which the robot chooses to participate are modeled. It is omnipotent and only its will is done. This has a number of consequences. First, it is impossible for it to coexist in a world with other intelligent entities. Since the robot expects only its will to be done, other entities engaging in processes affecting the world would totally disrupt the robot's model. Second, the almighty robot would clash with Mother Nature herself. She would maintain that grass can grow, that ice can melt, and that the sun can rise and set without the participation of the robot.

Another limiting factor common to many robot systems is the theory that action in the world is nothing more than a series of stills. This is analogous to the relationship which exists between a movie and a slide. A movie is a series of slides that changes so rapidly that it creates an illusion of continuous action.

No matter how many frames per second are used, there are still some unrecorded gaps. Within these gaps part of the original action is lost. A movie of falling snow does not tell a true story. The frame by frame analysis would seem to reveal that snow falls in a jerky fashion, going from one position, immediately to another position slightly lower, never having travelled the area in between. Because significant things can happen in these gaps, robot systems whose operators jump the model from one state to another at some set interval yield a false representation of the world causing failure in certain instances.

For example, if a bucket is to be *completely* filled from a tap without spilling, the moment the bucket is filled may occur within one of these gaps. Since operators can only be applied at these set intervals, the robot is unable to achieve the goal.

The world can be perceived as a collection of on-going processes. At any particular moment some processes will be completed and others will be initiated, but no one process can be initiated and completed at the same instant (excluding quantum theoretical effects). Each process, when viewed microscopically, takes place over a continuum through gradual change. Although from man's frame of reference a light seems to come on instantaneously, in actuality, there is a gradual increase in the brightness of its filament until its maximum level is reached. Moreover, these processes are of an interactive nature.

Any process may trigger the initiation or completion of some other process. The actions of each intelligent entity, including the robot itself, make up only a part of this total picture. Without these intelligent entities a great many processes would still maintain themselves.

This view of a process as "a bringing about of a set of changes through a continuum of alteration" is the basis of the system developed by Hendrix (1973). This system makes it possible to model simultaneous interactive processes and allows for such processes to be involuntarily activated.

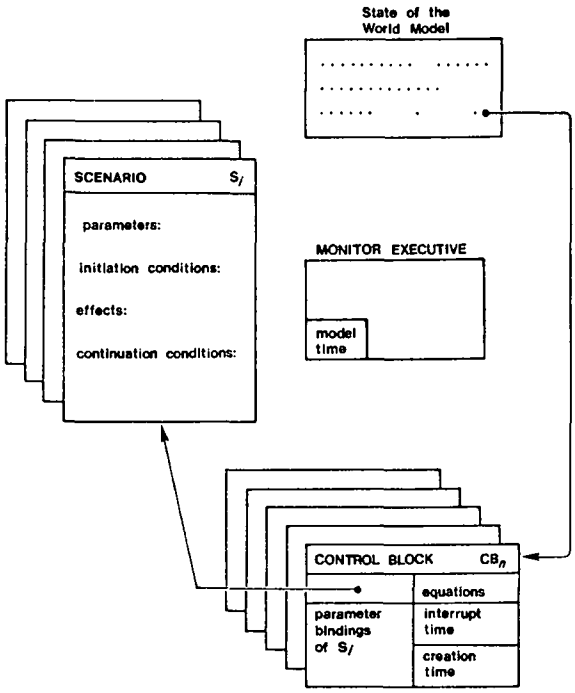


FIG. 1. Basic Hendrix simulator structure.

Like its predecessors, this system (Fig. 1) divides knowledge into state knowledge and process knowledge. State knowledge is maintained much as in other systems. An initial world model is presented as a list of characteristics. This list, referred to as the *state of the world model* (SWM), is continually updated. At any moment the SWM can be referenced to determine model world conditions.

In the earlier systems process knowledge is handled by a set of operators; in this system it is handled by a set of *scenarios*. Each is a description of one process. Each scenario contains three types of information. First, it contains the conditions which must exist for process initiation. Whenever the SWM satisfies these conditions the process begins. The second part of a scenario describes the process's effects on the SWM. This includes equations, which are dependent on time, to describe the gradual effects. The last part of a scenario is composed of continuation conditions. The process continues as long as the SWM satisfies these conditions. Each scenario describes the conditions which must exist for the process to begin and end plus its effects on the SWM.

Whenever a scenario's initiation conditions are satisfied a *control block* is created. The control block contains the relevant information representing an ongoing process. This information includes a reference to the scenario describing the process, the values satisfying that scenario's parameters, the equations found in that scenario with the appropriate parameter substitutions, the time the process was initiated, and the time (determined from the continuation conditions) when the process will end. The newly created control block is added to the control block list where it remains until the process it represents is terminated (or interrupted).

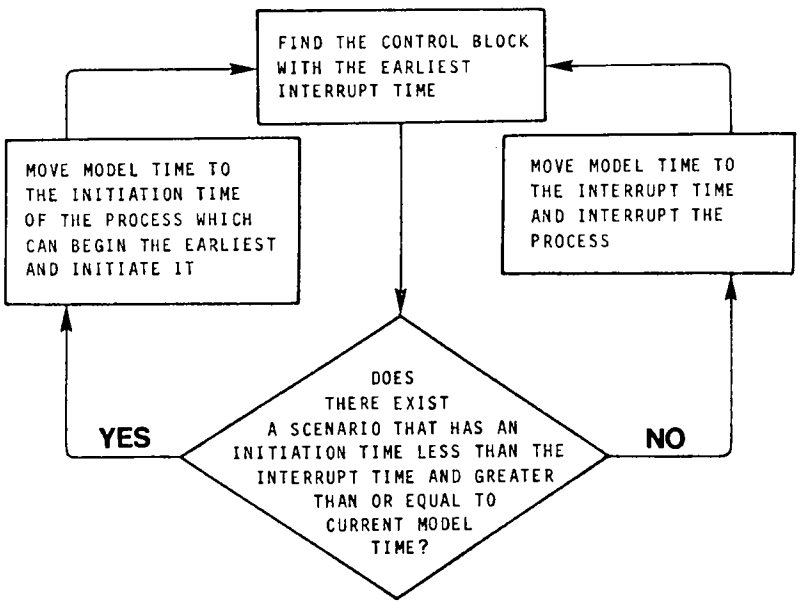


FIG. 2. Monitor executive flowchart.

The controling structure of the model is the *monitor executive* (Fig. 2). The monitor executive keeps model time, the scenario list describing all possible processes, and maintains the state of the world model. It also initiates, monitors, and interrupts processes when appropriate.

Model time is not kept by a conventional clock. Instead it is a real number that is initially set to zero and is then advanced as necessary by the monitor executive.

The monitor executive causes process initiation through the creation of a control block from the scenario describing the process. Alterations are made to the SWM as dictated by the effects portion of that scenario. A reference to the new control block is added to those SWM characteristics which are now described by the control block's equations. At any particular model time these equations can be solved to give the current value of those characteristics.

The monitor executive keeps track of all on-going processes by referencing the control block list since each control block represents an on-going process. When a process is interrupted or terminated its associated control block is destroyed.

The interruption or termination of a process takes place in the following way. All of the characteristics in the SWM which refer to the control block of the process about to be interrupted or terminated are altered. The equations in the control block that describe how these characteristics change with time are solved using the current model time. The solutions are substituted into the SWM. These alterations are made since the process responsible for the gradually changing nature of these characteristics has come to an end. The monitor executive then alters the SWM as is dictated by the effects portion of the scenario associated with the control block. The control block is then destroyed.

The monitor executive continually scans the model for an opportunity to initiate or interrupt a process. The monitor executive first determines which control block has the earliest interrupt time associated with it. If two or more control blocks have that interrupt time, the monitor chooses one arbitrarily. It then scans the scenario list looking for a scenario whose parameters can be bound such that the initiation conditions of that scenario are satisfied at an earlier time than the chosen interrupt time yet greater than or equal to current model time. If more than one such scenario is found, then the one with the earliest initiation time is chosen. The monitor executive then moves model time up to that time and initiates the process described by those bindings and that scenario. If no such bindings and scenario are found, the monitor executive moves model time up to that interrupt time and interrupts the process associated with that control block. In either case, after a process is initiated or interrupted the monitor executive goes through the same procedure looking for another process to affect. If more than one process can be initiated or interrupted at one time, then the monitor, functioning as described above, will hold model time still until all processes are affected as desired; those things which should happen at the same time do.

For any process to be initiated it is only necessary that the initiation conditions of the associated scenario be met. These conditions may exclude the robot, thereby allowing for processes in which the robot does not take part to be modeled.

All of the processes represented by the control blocks appear to modify the world simultaneously and at any moment in model time, the exact state of any process is known. By referencing the SWM and solving the related equations with the appropriate model time, the characteristics of the world can be obtained.

The remainder of the paper is composed of six sections. Section 2 includes three incrementally built robot worlds, each introducing a new aspect of modeling. Section 3 describes the interactive operating environment. Section 4 discusses three additional world models including a robot world (with some sample output), an electrical world, and a Turing world. A thorough discussion of our implementation is in section 5. Section 6 includes a discussion of the deviations in our implementation from the general model. Section 7 describes some of the finer points of modeling. The final subsection contains our concluding remarks.

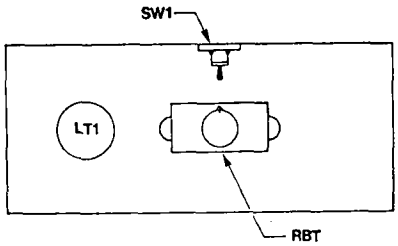
2. Modeling

2.1. AN INSTANTANEOUS MODEL

The first example world to be discussed is a very simple one. It consists of a robot, a switch and a light. The robot is permanently stationed in front of the switch. Its only task is to move the switch up or down when instructed. Since the switch controls the light, the light should come on whenever the switch is in the up position.

The modeling of this world begins with the formation of an initial state of the world model. This contains the components of the world and their relationship to one another. It may also include other relevant characteristics of the components. In this system the initial information is represented by a list of relations.

The construction of the initial state of the world model (Fig. 3) begins by assigning a symbolic name to each component. In this particular world the robot may be RBT, the switch SW1 and the light LT1. This information is represented by the relations (TYPE RBT ROBOT), (TYPE SW1 SWITCH) and (TYPE LT1 LIGHT). The only information needed regarding the relationship between these components is that the light is controlled by the switch. This is indicated by the relation (CONTROL SW1 LT1). Finally the other needed characteristics, that the switch is down and the light is off, are represented by (POSITION SW1 DOWN) and (STATE LT1 OFF). Other relations describing additional characteristics of the world could be included but are not since the above list of relations is sufficient to model the desired world.




```
(TYPE RBT ROBOT)

(TYPE LT1 LIGHT)
(STATE LT1 OFF)

(TYPE SW1 SWITCH)
(POSITION SW1 DOWN)
(CONTROL SW1 LT1)
```

FIG. 3. Initial World1 and SWM.

A list of scenarios, describing how the processes of the world change the components and their relationships, must also be included in the model. The processes of this world include the robot's operation of the switch and the resulting effects on the light. As with the initial state of the world model, a decision must be made concerning what should be included and what should not in process descriptions. Process descriptions are sufficient if they provide the needed effects on the SWM. This necessitates a choice of the *time frame* in which the modeling takes place.

Depending on the chosen time frame, the light's going on and off would be modeled quite differently. If a time frame is chosen which is close to a human's, the light should be modeled as if it goes on and off instantaneously. One generally does not perceive a light coming on as a process which takes place over a span of time. Depending on the application of the model it may be acceptable to adopt the time frame of humans and make the same assumptions. On the other hand, for modeling electron flow, a much different time frame must be chosen. These time frames may be chosen as coarse or fine as desired. The choice is model relative. It is usually easier to pick the coarsest level that still assures the desired effects. For this example, world electron flow is not of interest. Thus a time frame comparable to that on which humans function is adopted.

Processes in this world are modeled as if they were instantaneous. An instantaneous process is signified by the omission of that portion of a scenario which describes the duration of the process.

The first scenario (Fig. 4(a)) describes how a robot moves a switch up. The scenario begins with its *name*, SWITCHUP, and *parameter list* (PAR R S). The next part of the scenario is its *initiation conditions*. The process described by the scenario will begin when values can be substituted for the parameters, R and S, such that the initiation conditions are satisfied and no process need be affected at an earlier model time.

In SWITCHUP there are only *symbolic initiation conditions* (ICS). These are satisfied when the state of the world model contains these relations. The first is (TYPE R ROBOT). This guarantees that the value chosen for R is the symbolic name for a robot. The next ICS, (TYPE S SWITCH), guarantees that S is the name for a switch. The ICS (POSITION S DOWN) makes sure that switch S is down since only a switch which is down can be moved up. The next ICS, (ALLOCATE-ACTIVATE R SWITCHUP S), checks that the robot R has been instructed to move switch S up. The appearance of such an ALLOCATE-ACTIVATE relation in the state of the world model is an order to the robot.

The remainder of the scenario consists of the *initial effects*. Initial effects are those which take place immediately upon initiation of the process. They consist of relations that are to be deleted and added to the state of the world model. The values assigned to the parameters in order to satisfy the ICS are substituted for their respective parameters in the initial effects before they are used to affect the SWM.

The initial effects that are to be deleted (EID) are (POSITION S DOWN) and (ALLOCATE-ACTIVATE R SWITCHUP S). The first removes the fact that the switch is down and the second removes the ALLOCATE-ACTIVATE order since the order is being carried out. After these are deleted from the SWM, the other initial effects are added (EIA). In this case it is (POSITION S UP). This adds the fact that the switch is now up.

The scenario SWITCHUP can now be used by the system. Whenever the initiation conditions are satisfied the position of the switch in the SWM will be changed from down to up.

```
(SWITCHUP (PAR R S)
  (ICS (TYPE R ROBOT) (TYPE S SWITCH) (POSITION S DOWN)
    (ALLOCATE-ACTIVATE R SWITCHUP S))
  (EID (POSITION S DOWN) (ALLOCATE-ACTIVATE R SWITCHUP S))
  (EIA (POSITION S UP))      )
```

FIG. 4(a). World1 SWITCHUP scenario.

The next scenario, SWITCHDOWN (Fig. 4(b)), describes how the robot moves a switch down. It is analogous to SWITCHUP. The ALLOCATE-ACTIVATE order and the fact that the switch is on must be present for the process to start. The scenario causes the removal of the order and changes the position of the switch to down.

```
(SWITCHDOWN (PAR R S)
  (ICS (TYPE R ROBOT) (TYPE S SWITCH) (POSITION S UP)
    (ALLOCATE-ACTIVATE R SWITCHDOWN S))
  (EID (POSITION S UP) (ALLOCATE-ACTIVATE R SWITCHDOWN S))
  (EIA (POSITION S DOWN))      )
```

FIG. 4(b). World1 SWITCHDOWN scenario.

Now that the switch can be moved up and down, the light must be made to go on and off with the switch. This is accomplished by the scenarios LIGHTON and LIGHTOFF (Fig. 4(c)). LIGHTON has two parameters L and S. The ICS of the scenario guarantee that L is a light, that S is a switch, that switch S controls light L, and that the switch is up and the light off. This imbalance, that the switch is up yet the light is off, is the key thing that calls for the initiation of the process. Because a robot's participation is not needed to make lights go on and off with switches there is no robot included in the ICS. The initial effects simply correct the imbalance. LIGHTOFF is analogously defined.

```
(LIGHTON (PAR L S)
  (ICS (TYPE L LIGHT) (TYPE S SWITCH) (CONTROL S L) (POSITION S UP)
    (STATE L OFF))
  (EID (STATE L OFF))
  (EIA (STATE L ON))       )

(LIGHTOFF (PAR L S)
  (ICS (TYPE L LIGHT) (TYPE S SWITCH) (CONTROL S L)
    (POSITION S DOWN) (STATE L ON))
  (EID (STATE L ON))
  (EID (STATE L OFF))      )
```

FIG. 4(c). World1 LIGHTON and LIGHTOFF scenarios.

This completes the model of the world described. The robot can now be ordered by ALLOCATE–ACTIVATE relations to move the switch up and down. The light will respond as desired.

2.2. A CONTINUOUS MODEL

Now suppose that in addition to the light and switch there is a light that is controlled by a button. This new light is on while the button is depressed. The previous model can be updated to handle this new world.

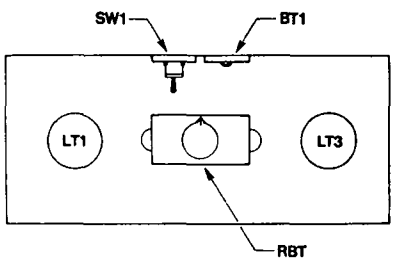
Additions must be made to the initial state of the world model (Fig. 5). The new information must include symbolic names for the button and light, that the button controls the light, that the button is currently out, and that the light is off. The relations (TYPE BT1 BUTTON), (TYPE LT3 LIGHT), (CONTROL BT1 LT3), (POSITION BT1 OUT) and (STATE LT3 OFF) convey this new information.

The process of the robot pushing the button can be handled by the scenario PUSHBUTTON (Fig. 6). PUSHBUTTON has two parameters R and B. The ICS of PUSHBUTTON are much the same as those for SWITCHUP and SWITCHDOWN. They guarantee that R is a robot, that B is a button, that R has been ordered to push button B, and that the button is not currently being pushed.

The initial effects of PUSHBUTTON differ from SWITCHUP and SWITCHDOWN only slightly. The order for the button to be pushed is not removed because the robot continues to push the button until the order is removed. Since the pushing of the button is an instantaneous act, *continuation conditions* must be included in the scenario.

The continuation conditions of this scenario are of the symbolic type (CCS). This signifies that the process is to continue as long as the relations in the CCS are in the state of the world model. The only CCS in PUSHBUTTON is the ALLOCATE–ACTIVATE order. So the process will continue as long as the order remains in the SWM.

When the process is interrupted by the deletion of the ALLOCATE–ACTIVATE order the *post effects* take place. Post effects are like initial effects except that they represent changes that occur in the world as a result of a process ending rather than beginning. They also are of two types, those which are to be deleted (EPD) and those which are to be added (EPA). The post effects of PUSHBUTTON delete the fact that the button is pushed and add that the button is out.




```
(TYPE RBT ROBOT)

(TYPE LT1 LIGHT)          (TYPE LT3 LIGHT)
(STATE LT1 OFF)           (STATE LT3 OFF)

(TYPE SW1 SWITCH)         (TYPE BT1 BUTTON)
(POSITION SW1 DOWN)       (POSITION BT1 OUT)
(CONTROL SW1 LT1)         (CONTROL BT1 LT3)
```

FIG. 5. Initial World2 and SWM.

BLIGHTON (Fig. 6) is the scenario which describes how the light connected to the button reacts when the button is pushed and released. The ICS guarantee that parameter L is a light and B a button, that button B controls light L, and that the button is in and the light off. This checks for an imbalance between the button and light much as LIGHTON and LIGHTOFF did. The initial effects correct the imbalance.

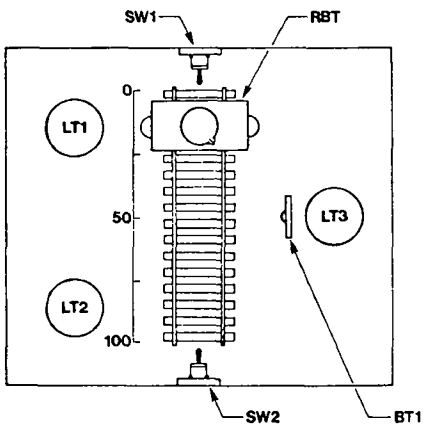
Since the light is to remain on while the button is being pushed, the CCS consists of one relation, that the button is in the IN position. When the button is released by the robot, the button's position changes to OUT and the process described by BLIGHTON ends. The post effects are the last to take place. They simply change the state of the light back to off.

```
(PUSHBUTTON (PAR R B)
  (ICS (TYPE R ROBOT) (TYPE B BUTTON)
    (ALLOCATE-ACTIVATE R PUSHBUTTON B) (POSITION B OUT))
  (EID (POSITION B OUT))
  (EIA (POSITION B IN))
  (CCS (ALLOCATE-ACTIVATE R PUSHBUTTON B))
  (EPD (POSITION B IN))
  (EPA (POSITION B OUT))      )

(BLIGHTON (PAR L B)
  (ICS (TYPE L LIGHT) (TYPE B BUTTON) (CONTROL B L) (POSITION B IN)
    (STATE L OFF))
  (EID (STATE L OFF))
  (EIA (STATE L ON))
  (CCS (POSITION B IN))                    Note: SWITCHUP, SWITCHDOWN,
  (EPD (STATE L ON))                           LIGHTON, and LIGHTOFF
  (EPA (STATE L OFF))      )                   are defined as before.
```

FIG. 6. World2_PUSHBUTTON and BLIGHTON scenarios.

2.3. A GRADUAL MODEL

In the previously discussed worlds the robot has been stationary. In the next world to be discussed (Fig. 7) the robot can move along a linear track that is 100 units long. There is one switch at each end of the track. Each switch controls a different light. At the halfway point on the track there is a button that controls a third light. The robot can only operate the switches and button when it is positioned directly in front of them. The robot responds to commands concerning moving switches up and down, pushing the button, and moving to a specified location.




```
(TYPE RBT ROBOT)
(AT RBT 0)
(RATE RBT 0)
(SPEEDLIMIT RBT 20)

(TYPE LT1 LIGHT)        (TYPE LT2 LIGHT)    (TYPE LT3 LIGHT)
(STATE LT1 OFF)         (STATE LT2 OFF)     (STATE LT3 OFF)

(TYPE SW1 SWITCH)       (TYPE SW2 SWITCH)
(POSITION SW1 DOWN)     (POSITION SW2 DOWN)
(CONTROL SW1 LT1)       (CONTROL SW2 LT2)
(AT SW1 0)              (AT SW2 100)

(TYPE BT1 BUTTON)
(POSITION BT1 OUT)
(CONTROL BT1 LT3)
(AT BT1 50)
```

FIG. 7. Initial World3 and SWM.

The initial state of the world model differs from the previous example only to reflect the robot's new mobility. The additional information includes the robot's current position, its speed, and the maximum rate at which it can travel. The robot begins at position 0. It is not moving, so its rate is also 0. Its maximum speed, expressed in units of length per unit of time, is 20. This information is expressed by (AT RBT 0), (RATE RBT 0) and (SPEEDLIMIT RBT 20). The information about lights LT1 and LT3 remains unchanged since their position is irrelevant. A new light, LT2, is added. The information about LT2 is expressed similarly to that of LT1. The positions of button BT1, switch SW1, and switch SW2 are expressed by the additional relations (AT BT1 50), (AT SW1) 0 and (AT SW2 100).

The scenarios LIGHTON, LIGHTOFF and BLIGHTON remain as they appeared in the previous models. No changes are necessary since lights go on and off with their respective switches and buttons regardless of locations. Since the robot must be able to reach a switch or button before affecting it, the scenarios SWITCHUP, SWITCHDOWN and PUSHBUTTON must be modified.

SWITCHUP (Fig. 8(a)) and SWITCHDOWN (Fig. 8(b)) are changed in the same way. A new parameter is added to their parameter lists. This parameter differs from those previously used in that its value must always be a number. This type of parameter is called a *C-variable*. It is denoted by the first letter of the parameter being a "C". The name C-variable comes from the fact that its value remains constant throughout the process. The particular C-variable used is CLOCATION. CLOCATION is separated from the other parameters by a colon. Parameters to the right of the colon are called *secondary parameters*. All others are *primary parameters* (the distinction between primary and secondary parameters is explained later). The ICS of the scenarios have (AT R CLOCATION) and (AT S CLOCATION) added to them. These are only satisfied when the robot R and the switch S are at the same place.

```
(SWITCHUP (PAR R S : CLOCATION)
  (ICS (TYPE R ROBOT) (TYPE S SWITCH) (POSITION S DOWN)
    (ALLOCATE-ACTIVATE R SWITCHUP S) (AT R CLOCATION)
    (AT S CLOCATION))
  (EID (POSITION S DOWN) (ALLOCATE-ACTIVATE R SWITCHUP S))
  (EIA (POSITION S UP))      )
```

FIG. 8(a). World3 SWITCHUP scenario.

```
(SWITCHDOWN (PAR R S : CLOCATION)
  (ICS (TYPE R ROBOT) (TYPE S SWITCH) (POSITION S UP)
    (ALLOCATE-ACTIVATE R SWITCHDOWN S) (AT R CLOCATION)
    (AT S CLOCATION))
  (EID (POSITION S UP) (ALLOCATE-ACTIVATE R SWITCHDOWN S))
  (EIA (POSITION S DOWN))      )
```

FIG. 8(b). World3 SWITCHDOWN scenario.

```
(PUSHBUTTON (PAR R B : CLOCATION)
  (ICS (TYPE R ROBOT) (TYPE B BUTTON)
    (ALLOCATE-ACTIVATE R PUSHBUTTON B) (POSITION B OUT)
    (AT R CLOCATION) (AT B CLOCATION))
  (EID (POSITION B OUT))
  (EIA (POSITION B IN))
  (CCS (ALLOCATE-ACTIVATE R PUSHBUTTON B) (AT R CLOCATION))
  (EPD (POSITION B IN))
  (EPA (POSITION B OUT))      )
```

FIG. 8(c). World3 PUSHBUTTON scenario.

PUSHBUTTON (Fig. 8(c)) is similarly changed. Like SWITCHUP and SWITCHDOWN the secondary parameter CLOCATION is added to the parameter list with (AT R CLOCATION) and (AT B CLOCATION) added to the ICS. But unlike the others, this necessitates an addition to the CCS. Since the robot can only continue to push the button if it remains in the same place (AT R CLOCATION) is added. If it were possible to move the button in this world (AT B CLOCATION) would also be needed, but since buttons in this world are stationary no such addition is made. There

are two conditions that can interrupt this process, the order being removed or the robot moving away from the button. If interruption comes from the robot's movement the order will remain. If the robot returns to the button at some later time the button may again be pushed.

One more scenario (the GOTO scenario) is needed to describe how the robot moves (Fig. 8(d)). R, CTO and CSPEED are the primary parameters and CFROM, CSPEEDLIMIT and ERATE are the secondary parameters. All of those that begin with "C" are C-variables and ERATE is an *E-variable*. The ICS differ somewhat from those used before. There is no (TYPE R ROBOT) as might have been expected. This does not appear since (SPEEDLIMIT R CSPEEDLIMIT) guarantees that R is a robot. This follows from the fact that only robots have speedlimits. Therefore (TYPE R ROBOT) would be redundant though it would work if included. Besides guaranteeing that R is a robot, it also gives the correct value to CSPEEDLIMIT. This value and the value for CFROM, which is bound by (AT R CFROM), are to be used later in the scenario.

In addition to symbolic initiation conditions, this scenario also has *numeric initiation conditions* (ICN). In order for the process to be initiated all ICN must hold. The ICN are restrictions on the C-variables. In this scenario the ICN guarantee that the place the robot is to go is on the track, that it is being told to go a legal speed greater than zero, and that it is not already there. These are (GE CTO 0), (LE CTO 100), (LE CSPEED CSPEEDLIMIT), (GT CSPEED 0) and (NOT(EQUAL CTO CFROM)), respectively. The first of each parenthesized set is a predicate. Any LISP (McCarthy, 1960) predicate can be used but the following are usually sufficient:

NOT
EQUAL
GT — greater than
LT — less than
GE — greater than or equal
LE — less than or equal.

Arithmetic functions are often used as arguments to the above predicates. A subset of these which is usually sufficient follows:

PLUS — the sum of two numbers
DIF — the difference of two numbers
TIMES — the product of two numbers
QUO — the quotient of two numbers
ABS — the absolute value of a number
ABDIF — the absolute difference of two numbers
SQ — the square of a number
SQRT — the square root of a number.

The last ICN (Fig. 8(d)) sets the E-variable ERATE to the positive speed if the robot is to move towards 100, and to the negative speed if the robot is to move towards 0. The ":=" is the assignment operator used to bind E-variables. E-variables behave like C-variables except that their values are calculated from C-variables. The assignment of an E-variable never affects the outcome of the ICN.

After the initiation conditions have been satisfied, the parameters are bound to the appropriate values and the initiation of the process can commence. The EID of GOTO contains one new feature, that is the appearance of "*". The EID (RATE R*) and (AT R*) causes the deletion of the relation describing the rate at which robot R is traveling, no matter what it is, and the deletion of the relation describing robot R's location.

Since the robot's location continually changes with time, the GOTO scenario has *gradual effects*. Gradual effects are of two types, symbolic (EGS) and numeric (EGN). The EGS are like other effects except that they contain at least one *Y-variable*. Y-variables always start with a "Y". A Y-variable has a numeric value that changes with respect to time. The way each Y-variable in the EGS varies is described by the equations in the EGN. Each relation in the EGS is added to the SWM without filling in any values for the Y-variables. At any time the EGN provide the information necessary to calculate the current values of the Y-variables.

```
(GOTO (PAR R CTO CSPEED : CFROM CSPEEDLIMIT ERATE)
  (ICS (ALLOCATE-ACTIVATE R GOTO CTO CSPEED) (AT R CFROM)
    (SPEEDLIMIT R CSPEEDLIMIT))
  (ICN (GE CTO O) (LE CTO 100) (LE CSPEED CSPEEDLIMIT) (GT CSPEED O)
    (NOT (EQUAL CTO CFROM))
    (:= ERATE (TIMES (QUO (DIF CTO CFROM)
                          (ABDIF CTO CFROM)) CSPEED))      )
  (EID (RATE R*) (AT R*))
  (EIA (RATE R ERATE))
  (EGS (AT R YPOSITION))
  (EGN ((:= YPOSITION (PLUS CFROM (TIMES ERATE $)))
        (:= $ (QUO (DIF YPOSITION CFROM) ERATE))  )    )
  (CCS (ALLOCATE-ACTIVATE R GOTO CTO CSPEED))
  (CCN (NOT (EQUAL CTO YPOSITION)))
  (EPD (RATE R*) (ALLOCATE-ACTIVATE R GOTO CTO CSPEED))
  (EPA (RATE R O))      )
```

FIG. 8(d). World3 GOTO scenario.

The EGN consists of equation pairs. The first of each pair uses := to assign the Y-variable an arithmetic expression which contains some *time-variable*. Time-variables include # and $. The variable # is the current model time and $ is the elapsed time since the process was initiated. In addition to one of these time-variables %, which represents the initiation time of the process, can also be used. The second equation in each pair is the same equation as the first only solved for # or $, whichever it contains. (The second equation is superflous and was included to ease the implementation.)

The gradual effects of GOTO describes how the robot moves with respect to time. The relation (AT R YPOSITION) from the EGS and the equations from the EGN relate that information. In normal arithmetic notation the equations in the EGN appear as YPOSITION = CFROM + (ERATE · $) and $ = (YPOSITION − CFROM) ÷ ERATE.

The CCS of GOTO are the same as those used before. It simply states that the process is to be interrupted if the ALLOCATE-ACTIVATE order is deleted from the SWM. If the order is removed the current location of the robot is calculated from the EGN and substituted into the SWM.

The last new feature in GOTO is the *numeric continuation conditions* (CCN). The only difference in appearance between the ICN and CCN is that the CCN contain Y-variables. These provide a means of checking the values of the Y-variables defined in the EGN. As long as the CCN hold the process is allowed to continue. But when one of them no longer holds, the process is interrupted. Thus interruption can result from the CCS or the CCN. In GOTO the CCN declare that the process is to continue as long as the robot has not yet reached its destination.

The post effects of GOTO present nothing new. They simply delete the rate at which the robot was traveling, add that its rate is now 0, and remove the order if it still exists.

```
(GOTO (PAR R CTO CSPEED : CFROM CSPEEDLIMIT ERATE)
  (ICS (ALLOCATE-ACTIVATE R GOTO CTO CSPEED) (AT R CFROM)
    (SPEEDLIMIT R CSPEEDLIMIT))
  (ICN (GE CTO O) (LE CTO 100) (LE CSPEED CSPEEDLIMIT) (GT CSPEED O)
    (NOT (EQUAL CTO CFROM))
    (:= ERATE (TIMES (QUO (DIF CTO CFROM)
                          (ABDIF CTO CFROM)) CSPEED))      )
  (EID (RATE R*) (AT R*))
  (EIA (RATE R ERATE))
  (EGS (AT R YPOSITION))
  (EGN ((:= YPOSITION (PLUS CFROM (TIMES ERATE $)))
        (:= $ (QUO (DIF YPOSITION CFROM) ERATE))  )    )
  (CCS (ALLOCATE-ACTIVATE R GOTO CTO CSPEED))
  (CCN FUNC (PLUS % (QUO (DIF CTO CFROM) ERATE))  )
  (EPD (RATE R*) (ALLOCATE-ACTIVATE R GOTO CTO CSPEED))
  (EPA (RATE R O))      )
```

FIG. 9. World3 functional GOTO scenario.

When the system uses a scenario which contains gradual effects it will construct equations from the EGN and the CCN that can then be solved simultaneously for the time at which the process will be completed. For GOTO this time is when the robot reaches its destination. Since the system does not contain a general equation solver, it only sets up the equations and then asks the user to solve them. The user is allowed to use LISP to help determine the solution. The solution is then typed in by the user and the system continues.

If the user is unwilling to solve the equations, there is an alternative. This alternative consists of giving the system an expression which will evaluate to the interrupt time. This expression is composed of any functions, but the aforementioned arithmetic functions are usually sufficient. This expression, prefixed by FUNC, is an alternative representation of the CCN. The system assumes that FUNC appearing in the CCN is followed by an expression which evaluates to the interrupt time.

The scenario GOTO (Fig. 9) can be modified to use a FUNC expression as its CCN. The expression states that the process will end at a model time equal to the time the process was initiated, %, plus the length to be traveled divided by the rate.

A good deal of power is derived from the fact that the FUNC expressions used in the CCN and those used as the second arguments to := in the EGN can be any LISP expression. Specifically, numerical methods can be employed to seek approximations. The square root function, SQRT, is an example of such a function. The largest acceptable error value is bound to EPSILON. EPSILON is discussed later.

It was mentioned earlier that those parameters in a scenario's parameter list which are to the right of a colon are secondary parameters and all others are primary parameters. These are instrumental in preventing the same process from being modeled more than once. This is a problem since the initiation conditions of a scenario like GOTO are satisfied for the duration of the process. At any time during the movement of the robot the initiation conditions of GOTO are satisfied such that the same process of the robot going to the same place can be initiated. These additional initiations of the same process must be inhibited in order to prevent infinitely many initiations of the same process. This inhibition is accomplished by prohibiting any initiations of processes which use the same scenario and have the same primary parameter bindings.

This completes the discussion of this example world except for noting that these scenarios are written in a general way. With these scenarios an arbitrary number of robots, switches, buttons, and lights can be modeled without any changes to the scenarios; the modeler must take responsibility for guarding against unforeseen secondary effects.

3. Operation of the system

This implementation of the Hendrix simulator system is designed for an interactive terminal connected to a DECsystem-10 equipped with STANFORD A.I. LISP 1.6 (Quam, 1969). Once in LISP, the simulator is started by evaluating the function HSIM with no arguments. The system then responds with a request for a list of scenarios. The user has the option of typing in the scenarios or responding with the evaluation of some expression. If the second option is used, the function EVAL is listed with the expression to be evaluated. The expression is usually a LISP atom which has been set earlier to a list of scenarios. Either way the system comes back with the name of all the scenarios just input.

The system then requests a list of relations to be used as the initial state of the world model. Again these may be typed in or an expression can be evaluated as before.

After the scenarios and the SWM have been input the word COMMAND is printed, signifying to the user that the system is at the command level. At command level the user can affect and inspect the system in a number of ways. The user types in the command desired and the system executes it and returns to command level. The only exception is the command *STOP*. It terminates the run and prints out the model time at which the termination occurred.

The commands *TIME*, *SCENARIOS*, *ACTION* and *PICTURE* are all used to inspect the system. TIME prints the current model time. SCENARIOS responds with the name of all current scenarios in the system. ACTION prints the scenario names and the parameter bindings of each control block currently existing. PICTURE prints out the current state of the world model. This includes model time, the *explicit relations* and the *skeleton relations* (not to be confused with relation skeletons, Hendrix, 1973) solved at the current time. The explicit relations are those relations in the SWM which contain no Y-variables. The skeleton relations are those which do contain Y-variables. For each Y-variable the associated equation is found and solved. These values for the Y-variables are substituted into the skeleton relations before being printed. PICTURE labels those relations which are explicit and those which are skeletons so that the user will know which are changing. Normally there will be no skeleton relations included in the picture

because command level is usually only achieved after the completion of all gradual processes (discussed in section 5).

There are two commands which affect the state of the world model. These are *ADD* and *DEL*. ADD is listed with any number of relations. That is, the user enters (ADD relation_1 relation_2 . . . relation_k). These relations are then added to the explicit relations in the SWM. Similarly, DEL is used to remove relations from the SWM.

ADDSC is used to add a new scenario. ADDSC is listed along with the scenario definition. The system then adds this new scenario to the scenario list.

GO is the command which starts the modeling process. It starts the monitor executive looking for processes to initiate and interrupt. As long as there remains a process which can be initiated or interrupted before model time infinity, the monitor executive will continue to move model time up to these critical points and initiate or interrupt the processes. This necessarily means that all processes which have an interrupt time less than infinity will have been completed before the monitor executive allows the system to go back to command level.

The command level can be achieved earlier by using *BREAK*. When BREAK is listed with a model time, a break point is set for that time. That is, when model time is equal to the break point the action of the monitor executive is suspended and control is passed to the command level. While at this break point command level all commands except STOP are functional. The monitor executive will resume where it left off after the GO command is given.

Other commands allow the user to inspect the system while in motion without having to set break points. One such command is *SNAPSHOT:* SNAPSHOT prints the SWM just as PICTURE does only at prearranged times. SNAPSHOT is listed with future model times. Whenever the model time is about to be moved past one of these snapshot times, model time is first moved to that snapshot time and a picture is taken. If the user does not know what the interesting snapshot times will be, he can use *AUTOSNAP*. AUTOSNAP takes a snapshot at each critical point. This allows the user to see the SWM at all times involving the initiation or interruption of a process. The command *UNSNAP* turns this feature off.

Another command useful for inspecting the system while in motion is *TRACE*. TRACE informs the user about the creation and destruction of control blocks while the system is running. The time of the creation or destruction is printed along with the name of the scenario and the parameter bindings associated with that control block. TRACE is simply listed along with the scenario names the user wishes to have traced. If * is used instead of scenario names, all scenarios will be traced. Until a user is very comfortable with the system it is probably best that the TRACE with * always be used. The tracing is curtailed by listing *UNTRACE* with the desired scenario names. All tracing is stopped if * is used in place of scenario names.

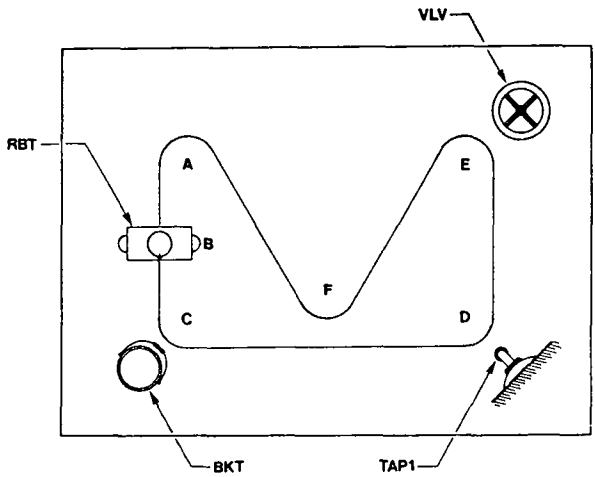
The command *EPSILON* is particularly different from the others. It exists because of the inherent errors made in calculations by digital computers. It is designed to compensate for these errors. EPSILON is listed with a number that becomes the largest difference that can exist between two numbers in the system and still be considered equal. This is necessary since small errors built up through successive calculations can prevent the initiation of processes that have initiation conditions requiring equal numbers. EPSILON is also useful as the maximum error allowable in any numerical methods being used. The system initially sets EPSILON to 0.000001.

4. Additional model worlds

The following worlds primarily demonstrate the versatility of the simulation system. The first example world is another gradual robot world. The other model worlds are instantaneous non-robotic worlds. An unconventional use of E-variables is introduced in the last world.

4.1. THE HENDRIX WORLD

This example world is the one Hendrix (1974) used in connection with GROPE (Friedman, 1973). The world (Fig. 10) includes a robot that moves on a track between locations A, B, C, D, E and F. The robot can only move forward alphabetically (also F to A allowing the robot to move in a cycle). There is a bucket, a valve and a tap. The




```
(TYPE A LOCATION)          (PATH A B 10.0)
(TYPE B LOCATION)          (PATH B C 10.0)
(TYPE C LOCATION)          (PATH C D 40.0)
(TYPE D LOCATION)          (PATH D E 20.0)
(TYPE E LOCATION)          (PATH E F 35.0)
(TYPE F LOCATION)          (PATH F A 35.0)

(TYPE RBT ROBOT)           (TYPE TAP1 TAP)
(AT RBT B)                 (AT TAP1 D)
(MOVABLE RBT)              (IMMOVABLE TAP1)
                           (GRASPABLE TAP1)
(TYPE BKT BUCKET)          (NOTGRASPED TAP1)
(AT BKT C)
(MOVABLE BKT)
(GRASPABLE BKT)
(NOTGRASPED BKT)
(CAPACITY BKT 100)
(CONTENT BKT 0)

(TYPE VLV VALVE)
(AT VLV E)
(IMMOVABLE VLV)
(GRASPABLE VLV)
(NOTGRASPED VLV)
(CONTROL VLV TAP1)
(MAXRATE VLV 10)
(RATE VLV 0)
```

FIG. 10. Initial WorldH and SWM.

```
(GRASP (PAR R OBJ : LOC)
  (ICS (ALLOCATE-ACTIVATE R GRASP OBJ) (GRASPABLE OBJ)
    (NOTGRASPED OBJ) (AT R LOC) (AT OBJ LOC))
  (EID (NOTGRASPED OBJ) (ALLOCATE-ACTIVATE R GRASP OBJ))
  (EIA (GRASPING R OBJ))       )

(RELEASE (PAR R OBJ : LOC)
  (ICS (ALLOCATE-ACTIVATE R RELEASE OBJ) (GRASPING R OBJ) (AT R LOC))
  (EID (ALLOCATE-ACTIVATE R RELEASE OBJ) (GRASPING R OBJ))
  (EIA (NOTGRASPING OBJ))       )

(TURNVALVE (PAR R V CRATE : CMAXRATE LOC)
  (ICS (ALLOCATE-ACTIVATE R TURNVALVE V CRATE) (AT R LOC) (AT V LOC)
    (MAXRATE V CMAXRATE))
  (ICN (LE CRATE CMAXRATE) (GE CRATE 0))
  (EID (RATE V *) (ALLOCATE-ACTIVATE R TURNVALVE V CRATE))
  (EIA (RATE V CRATE))       )

(MOVABILITY (PAR R : OBJ)
  (ICS (GRASPING R OBJ) (IMMOVABLE OBJ))
  (EID (MOVABLE R))
  (EIA (IMMOVABLE R))
  (CCS (GRASPING R OBJ) (IMMOVABLE OBJ))
  (EPD (IMMOVABLE R))
  (EPA (MOVABLE R))       )

(GOTO (PAR R PTO : CDIST PFROM)
  (ICS (ALLOCATE-ACTIVATE R GOTO PTO) (MOVABLE R) (AT R PFROM)
    (PATH PFROM PTO CDIST))
  (EID (AT R PFROM))
  (EIA (MOVING R PFROM PTO))
  (EGS (BETWEEN R YFRACTION PFROM PTO))
  (EGN((:= YFRACTION (QUO (TIMES 10 $) CDIST))
       (:= $ (QUO (TIMES YFRACTION CDIST) 10))  )  )
  (CCN FUNC (PLUS (TIMES 0.1 CDIST) %))
  (EPD (ALLOCATE-ACTIVATE R GOTO PTO) (BETWEEN R * * *) (MOVING R * *))
  (EPA (AT R PTO))       )

(MOVE (PAR OBJ : PFROM PTO CDIST)
  (ICS (MOVING R PFROM PTO) (GRASPING R OBJ) (AT OBJ PFROM)
    (PATH PFROM PTO CDIST))
  (EID (AT OBJ PFROM))
  (EIA (MOVING OBJ PFROM PTO))
  (EGS (BETWEEN OBJ YFRACTION PFROM PTO))
  (EGN ((:= YFRACTION (QUO (TIMES 10 $) CDIST))))
       (:= $ (QUO (TIMES 10 $) CDIST))
  (CCN FUNC (PLUS (TIMES 0.1 CDIST) %))
  (EPD (BETWEEN OBJ * * *) (MOVING OBJ * *))
  (EPA (AT OBJ PTO))       )

(FILLBUCKET (PAR B TP : V LOC CRATE CCAPACITY CINITIALCONTENT)
  (ICS (AT B LOC) (AT TP LOC) (CONTROL V TP) (RATE V CRATE)
    (CONTENT B CINITIALCONTENT) (CAPACITY B CCAPACITY))
  (ICN (GT CRATE 0) (LT CINITIALCONTENT CCAPACITY))
  (EID (CONTENT B *))
  (EGS (CONTENT B YCONTENT))
  (EGN ((:= YCONTENT (PLUS (TIMES CRATE $) CINITIALCONTENT))
       (:= $ (QUO (DIF YCONTENT CINITIALCONTENT) CRATE))  )  )
  (CCS (RATE V CRATE) (AT B LOC))
  (CCN FUNC (PLUS % (QUO (DIF CCAPACITY CINITIALCONTENT)) CRATE))  )
```

FIG. 11. WorldH scenarios.

tap and valve are stationary at positions D and E, respectively. The valve controls the tap. If the bucket is under the tap when the valve is open, the bucket will fill. The robot may grasp those objects which are graspable. If the robot moves while grasping a movable object the object moves along with it. If the robot attempts to move while grasping an immovable object, it fails to move.

The scenario GRASP (Fig. 11) allows the robot to grasp graspable objects when the object and robot are at the same location. RELEASE allows the robot to release an object while at one of the model locations. When the robot is at the valve, TURNVALVE allows the robot to turn the valve achieving the desired rate of flow from the tap. MOVABILITY determines when the robot is unable to move. GOTO allows the robot to move when it is able. MOVE monitors the movement of objects grasped by the robot as the robot moves. FILLBUCKET monitors the filling of the bucket when it is under the flowing tap. These scenarios are direct translations of those used by Hendrix (1974) (except that his scenarios do not remove the ALLOCATE–ACTIVATE order upon completion of the task).

A sample run of this model follows. We encourage the reader to scrutinize the output to aid comprehension of scenario interactions within the model. The atoms SLIST and SWM are bound to the list of scenarios and the list of explicit relations, respectively. This run also includes scenarios describing an alarm clock (discussed in section 7) which the robot can manipulate.

WorldH

```
◆⟨HSIM⟩

==============================
HENDRIX SIMULATING SYSTEM
==============================

INPUT SCENARIO LIST: ◆⟨EVAL SLIST⟩

GRASP RELEASE TURNVALVE MONITORSET SETALARM MONITORALARM SOUNDALARM O
FFALARM1 OFFALARM2 MOVABILITY GOTO MOVE FILLBUCKET

INPUT SWM RELATION LIST: ◆⟨EVAL SWM⟩


COMMAND: ◆⟨TRACE ◆⟩


COMMAND: ◆PICTURE


◆◆◆◆TIME◆◆◆◆
0.
◆◆◆◆EXPRS◆◆◆◆
⟨PATH A B 10.0⟩
⟨PATH B C 10.0⟩
⟨PATH C D 40.0⟩
⟨PATH D E 20.0⟩
⟨PATH E F 35.0⟩
⟨PATH F A 35.0⟩
⟨TYPE A LOCATION⟩
⟨TYPE B LOCATION⟩
⟨TYPE C LOCATION⟩
⟨TYPE D LOCATION⟩
⟨TYPE E LOCATION⟩
⟨TYPE F LOCATION⟩
⟨TYPE CLK CLOCK⟩
⟨TYPE RBT ROBOT⟩
⟨TYPE BKT BUCKET⟩
⟨TYPE VLV VALVE⟩
⟨TYPE TAP1 TAP⟩
⟨MOVABLE RBT⟩
⟨MOVABLE CLK⟩
⟨MOVABLE BKT⟩
⟨IMMOVABLE VLV⟩
⟨IMMOVABLE TAP1⟩
```

```
(GRASPABLE CLK)
(GRASPABLE BKT)
(GRASPABLE TAP1)
(GRASPABLE VLV)
(NOTGRASPED CLK)
(NOTGRASPED BKT)
(NOTGRASPED VLV)
(NOTGRASPED TAP1)
(AT CLK A)
(AT RBT B)
(AT BKT C)
(AT VLV E)
(AT TAP1 D)
(ALARM OFF CLK)
(CAPACITY BKT 100.)
(CONTENT BKT 0.)
(CONTROL VLV TAP1)
(MAXRATE VLV 10.)
(RATE VLV 0.)
****SKLPS****
**********

COMMAND: *(ADD(ALLOCATE-ACTIVATE RBT GOTO C)
*(ALLOCATE-ACTIVATE RBT GRASP BKT)
*(ALLOCATE-ACTIVATE RBT GOTO D))


COMMAND: *GO


<<<CREATING CB>>> TIME = 0.
GOTO ** (R RBT) (PTO C) (CDIST 10.0) (PFROM B)

<<<DESTROYING CB>>> TIME = 1.0
GOTO ** (R RBT) (PTO C) (CDIST 10.0) (PFROM B)

<<<CREATING CB>>> TIME = 1.0
GRASP ** (R RBT) (OBJ BKT) (LOC C)

<<<DESTROYING CB>>> TIME = 1.0
GRASP ** (R RBT) (OBJ BKT) (LOC C)

<<<CREATING CB>>> TIME = 1.0
GOTO ** (R RBT) (PTO D) (CDIST 40.0) (PFROM C)

<<<CREATING CB>>> TIME = 1.0
MOVE ** (R RBT) (OBJ BKT) (PFROM C) (PTO D) (CDIST 40.0)

<<<DESTROYING CB>>> TIME = 5.0
MOVE ** (R RBT) (OBJ BKT) (PFROM C) (PTO D) (CDIST 40.0)

<<<DESTROYING CB>>> TIME = 5.0
GOTO ** (R RBT) (PTO D) (CDIST 40.0) (PFROM C)
```

```
COMMAND: ◆PICTURE


◆◆◆◆TIME◆◆◆◆
5.0
◆◆◆◆EXPRS◆◆◆◆
(GRASPING RBT BKT)
(PATH A B 10.0)
(PATH B C 10.0)
(PATH C D 40.0)
(PATH D E 20.0)
(PATH E F 35.0)
(PATH F A 35.0)
(TYPE A LOCATION)
(TYPE B LOCATION)
(TYPE C LOCATION)
(TYPE D LOCATION)
(TYPE E LOCATION)
(TYPE F LOCATION)
(TYPE CLK CLOCK)
(TYPE RBT ROBOT)
(TYPE BKT BUCKET)
(TYPE VLV VALVE)
(TYPE TAP1 TAP)
(MOVABLE RBT)
(MOVABLE CLK)
(MOVABLE BKT)
(IMMOVABLE VLV)
(IMMOVABLE TAP1)
(GRASPABLE CLK)
(GRASPABLE BKT)
(GRASPABLE TAP1)
(GRASPABLE VLV)
(NOTGRASPED CLK)
(NOTGRASPED VLV)
(NOTGRASPED TAP1)
(AT RBT D)
(AT BKT D)
(AT CLK A)
(AT VLV E)
(AT TAP1 D)
(ALARM OFF CLK)
(CAPACITY BKT 100.)
(CONTENT BKT 0.)
(CONTROL VLV TAP1)
(MAXRATE VLV 10.)
(RATE VLV 0.)
◆◆◆◆SKLRS◆◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆

COMMAND: ◆(ADD (ALLOCATE-ACTIVATE RBT RELEASE BKT)
◆(ALLOCATE-ACTIVATE RBT GOTO E)
◆(ALLOCATE-ACTIVATE RBT TURNVALVE VLV 1.25)
◆(ALLOCATE-ACTIVATE RBT GOTO F)
◆(ALLOCATE-ACTIVATE RBT GOTO A)
◆(ALLOCATE-ACTIVATE RBT SETALARM CLK 20))

COMMAND: ◆(BREAK 21)
```

```
COMMAND: ♦(SNAPSHOT 20.5)

COMMAND: ♦GO

<<<CREATING CB>>> TIME = 5.0
RELEASE ♦♦ (R RBT) (OBJ BKT) (LOC D)

<<<DESTROYING CB>>> TIME = 5.0
RELEASE ♦♦ (R RBT) (OBJ BKT) (LOC D)

<<<CREATING CB>>> TIME = 5.0
GOTO ♦♦ (R RBT) (PTO E) (CDIST 20.0) (PFROM D)

<<<DESTROYING CB>>> TIME = 7.0
GOTO ♦♦ (R RBT) (PTO E) (CDIST 20.0) (PFROM D)

<<<CREATING CB>>> TIME = 7.0
TURNVALVE ♦♦ (R RBT) (V VLV) (CRATE 1.25) (CMAXRATE 10.) (LOC E)

<<<DESTROYING CB>>> TIME = 7.0
TURNVALVE ♦♦ (R RBT) (V VLV) (CRATE 1.25) (CMAXRATE 10.) (LOC E)

<<<CREATING CB>>> TIME = 7.0
GOTO ♦♦ (R RBT) (PTO F) (CDIST 35.0) (PFROM E)

<<<CREATING CB>>> TIME = 7.0
FILLBUCKET ♦♦ (B BKT) (TP TAP1) (V VLV) (LOC D) (CRATE 1.25) (CCAPACI
TY 100.) (CINITIALCONTENT 0.)

<<<DESTROYING CB>>> TIME = 10.5
GOTO ♦♦ (R RBT) (PTO F) (CDIST 35.0) (PFROM E)

<<<CREATING CB>>> TIME = 10.5
GOTO ♦♦ (R RBT) (PTO A) (CDIST 35.0) (PFROM F)

<<<DESTROYING CB>>> TIME = 14.0
GOTO ♦♦ (R RBT) (PTO A) (CDIST 35.0) (PFROM F)

<<<CREATING CB>>> TIME = 14.0
SETALARM ♦♦ (R RBT) (KLOCK CLK) (CSTIME 20.) (LOC A)

<<<DESTROYING CB>>> TIME = 14.0
SETALARM ♦♦ (R RBT) (KLOCK CLK) (CSTIME 20.) (LOC A)

<<<CREATING CB>>> TIME = 14.0
MONITORALARM ♦♦ (KLOCK CLK) (CSTIME 20.)

<<<DESTROYING CB>>> TIME = 20.
MONITORALARM ♦♦ (KLOCK CLK) (CSTIME 20.)

<<<CREATING CB>>> TIME = 20.
SOUNDALARM ♦♦ (KLOCK CLK) (CSTIME 20.)

<<<DESTROYING CB>>> TIME = 20.
SOUNDALARM ♦♦ (KLOCK CLK) (CSTIME 20.)
```

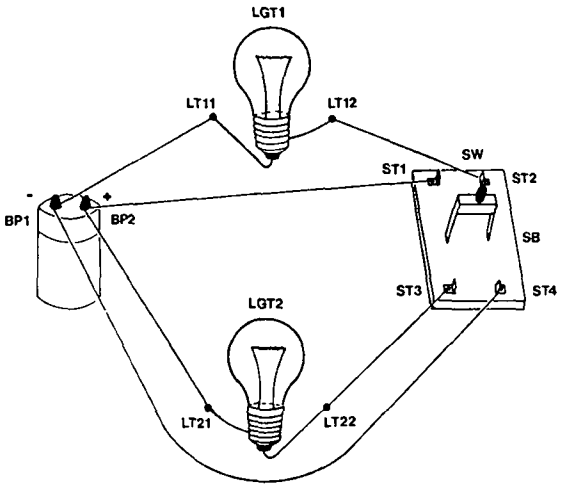
```
◆◆◆◆TIME◆◆◆◆
20.5
◆◆◆◆EXPRS◆◆◆◆
(PATH A B 10.0)
(PATH B C 10.0)
(PATH C D 40.0)
(PATH D E 20.0)
(PATH E F 35.0)
(PATH F A 35.0)
(TYPE A LOCATION)
(TYPE B LOCATION)
(TYPE C LOCATION)
(TYPE D LOCATION)
(TYPE E LOCATION)
(TYPE F LOCATION)
(TYPE CLK CLOCK)
(TYPE RBT ROBOT)
(TYPE BKT BUCKET)
(TYPE VLV VALVE)
(TYPE TAP1 TAP)
(MOVABLE RBT)
(MOVABLE CLK)
(MOVABLE BKT)
(IMMOVABLE VLV)
(IMMOVABLE TAP1)
(GRASPABLE CLK)
(GRASPABLE BKT)
(GRASPABLE TAP1)
(GRASPABLE VLV)
(NOTGRASPED BKT)
(NOTGRASPED CLK)
(NOTGRASPED VLV)
(NOTGRASPED TAP1)
(AT RBT A)
(AT BKT D)
(AT CLK A)
(AT VLV E)
(AT TAP1 D)
(ALARM SOUNDING CLK)
(CAPACITY BKT 100.)
(CONTROL VLV TAP1)
(MAXRATE VLV 10.)
(RATE VLV 1.25)
◆◆◆◆SKLRS◆◆◆◆
(CONTENT BKT 16.875)
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
```

```
==============================
>>>BREAK AT 21.
==============================

COMMAND: ♦(ADD(ALLOCATE-ACTIVATE RBT GOTO B)
♦(ALLOCATE-ACTIVATE RBT GOTO C)
♦(ALLOCATE-ACTIVATE RBT GOTO D)
♦(ALLOCATE-ACTIVATE RBT GOTO E)
♦(ALLOCATE-ACTIVATE RBT TURNVALVE VLV 0))


COMMAND: ♦GO

==============================

<<<CREATING CB>>> TIME = 21.
GOTO ♦♦ (R RBT) (PTO B) (CDIST 10.0) (PFROM A)

<<<DESTROYING CB>>> TIME = 22.0
GOTO ♦♦ (R RBT) (PTO B) (CDIST 10.0) (PFROM A)

<<<CREATING CB>>> TIME = 22.0
GOTO ♦♦ (R RBT) (PTO C) (CDIST 10.0) (PFROM B)

<<<DESTROYING CB>>> TIME = 23.0
GOTO ♦♦ (R RBT) (PTO C) (CDIST 10.0) (PFROM B)

<<<CREATING CB>>> TIME = 23.0
GOTO ♦♦ (R RBT) (PTO D) (CDIST 40.0) (PFROM C)

<<<DESTROYING CB>>> TIME = 27.0
GOTO ♦♦ (R RBT) (PTO D) (CDIST 40.0) (PFROM C)

<<<CREATING CB>>> TIME = 27.0
GOTO ♦♦ (R RBT) (PTO E) (CDIST 20.0) (PFROM D)

<<<DESTROYING CB>>> TIME = 29.0
GOTO ♦♦ (R RBT) (PTO E) (CDIST 20.0) (PFROM D)

<<<DESTROYING CB>>> TIME = 29.0
FILLBUCKET ♦♦ (B BKT) (TP TAP1) (V VLV) (LOC D) (CRATE 1.25) (CCAPACI
TY 100.) (CINITIALCONTENT 0.)

<<<CREATING CB>>> TIME = 29.0
TURNVALVE ♦♦ (R RBT) (V VLV) (CRATE 0.) (CMAXRATE 10.) (LOC E)

<<<DESTROYING CB>>> TIME = 29.0
TURNVALVE ♦♦ (R RBT) (V VLV) (CRATE 0.) (CMAXRATE 10.) (LOC E)
```

```
COMMAND: ♦PICTURE


♦♦♦♦TIME♦♦♦♦
29.0
♦♦♦♦EXPRS♦♦♦♦
(PATH A B 10.0)
(PATH B C 10.0)
(PATH C D 40.0)
(PATH D E 20.0)
(PATH E F 35.0)
(PATH F A 35.0)
(TYPE A LOCATION)
(TYPE B LOCATION)
(TYPE C LOCATION)
(TYPE D LOCATION)
(TYPE E LOCATION)
(TYPE F LOCATION)
(TYPE CLK CLOCK)
(TYPE PBT POBOT)
(TYPE BKT BUCKET)
(TYPE VLV VALVE)
(TYPE TAP1 TAP)
(MOVABLE RBT)
(MOVABLE CLK)
(MOVABLE BKT)
(IMMOVABLE VLV)
(IMMOVABLE TAP1)
(GRASPABLE CLK)
(GRASPABLE BKT)
(GRASPABLE TAP1)
(GRASPABLE VLV)
(NOTGRASPED BKT)
(NOTGRASPED CLK)
(NOTGRASPED VLV)
(NOTGRASPED TAP1)
(AT PBT E)
(AT BKT D)
(AT CLK A)
(AT VLV E)
(AT TAP1 D)
(ALARM SOUNDING CLK)
(CAPACITY BKT 100.)
(CONTENT BKT 27.5)
(CONTROL VLV TAP1)
(MAXRATE VLV 10.)
(RATE VLV 0.)
♦♦♦♦SKLRS♦♦♦♦
♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦

COMMAND: ♦STOP

(♦♦♦♦TERMINATED-AT-TIME♦♦♦♦ 29.0)
```

4.2. THE ELECTRIC WORLD

This model world (Fig. 12) consists of a battery, two lights and a switch. The switch has three positions, center, up and down. When the switch is up it connects the upper two switch terminals; when it is down it connects the lower two switch terminals; and when it is in the center position none of the terminals are connected.




```
(TYPE LGT1 LIGHT)          (TYPE LGT2 LIGHT)
(STATE LGT1 OFF)           (STATE LGT2 OFF)
(CIRCUIT LGT1 C1)          (CIRCUIT LGT2 C2)
(OBJ LGT1 RESISTANCE)      (OBJ LGT2 RESISTANCE)
(TERMINAL LGT1 LT11)       (TERMINAL LGT2 LT21)
(TERMINAL LGT1 LT12)       (TERMINAL LGT2 LT22)

(TYPE SW SWITCH)           (TERMINAL ST1 SW)
(SWITCHBLADE SB SW)        (TERMINAL ST2 SW)
(POSITION CENTER SB)       (TERMINAL ST3 SW)
(PPOSITION UP SB)          (TERMINAL ST4 SW)
(PPOSITION DOWN SB)        (SLOC ST1 ST2 UP)
                           (SLOC ST3 ST4 DOWN)

(VOLTCON BP1 DIR NEG)      (END BP1 LINE1)
(VOLTCON LT11 DIR NEG)     (END LT11 LINE1)
(VOLTCON LT12 RES NEG)     (END LT12 LINE2)
(VOLTCON ST2 RES NEG)      (END ST2 LINE2)
(VOLTCON BP2 DIR POS)      (END BP2 LINE3)
(VOLTCON ST1 DIR POS)      (END ST1 LINE3)
                           (END BP2 LINE4)
(VOLTCON LT21 DIR POS)     (END LT21 LINE4)
(VOLTCON LT22 RES POS)     (END LT22 LINE5)
(VOLTCON ST3 RES POS)      (END ST3 LINE5)
                           (END BP1 LINE6)
(VOLTCON ST4 DIR NEG)      (END ST4 LINE6)
```

FIG. 12. Initial WorldE and SWM.

A more detailed perspective is taken concerning the lights going on and off than in the previous models. Yet this does not represent an attempt to model general *RLC* electrical networks. In this model a light is on if it is part of a circuit with current. LIGHTON is the scenario that describes this. Current is produced in a circuit when an object of resistance contained in the circuit has one terminal directly connected to a positive terminal of some power source and the other directly connected to a negative terminal of some power source. This is handled by scenario CURRENT.

```
(LIGHTON (PAR L : LCIR)
  (ICS (CURRENT LCIR) (CIRCUIT L LCIR) (TYPE L LIGHT))
  (EID (STATE L OFF))
  (EIA (STATE L ON))
  (CCS (CURRENT LCIR))
  (EPD (STATE L ON))
  (EPA (STATE L OFF))       )

(CURRENT (PAR RCIR : T1 T2 OBJR)
  (ICS (VOLTCON T1 DIR POS) (VOLTCON T2 DIR NEG) (TERMINAL OBJR T1)
    (TERMINAL OBJR T2) (CIRCUIT OBJR RCIR) (OBJ OBJR RESISTANCE))
  (EIA (CURRENT RCIR))
  (CCS (VOLTCON T1 DIR POS) (VOLTCON T2 DIR NEG))
  (EPD (CURRENT RCIR))       )

(MOVE (PAR S POS : B NPOS)
  (ICS (MOVE S POS) (POSITION NPOS B) (PPOSITION POS B)
    (SWITCHBLADE B S))
  (EID (POSITION NPOS B) (PPOSITION POS B) (MOVE S POS))
  (EIA (POSITION POS B) (PPOSITION NPOS B))       )

(SWITCH (PAR S POS : T1 T2 B)
  (ICS (SLOC T1 T2 POS) (POSITION POS B) (TERMINAL T1 S)
    (TERMINAL T2 S) (SWITCHBLADE B S))
  (EIA (END T1 B) (END T2 B))
  (CCS (POSITION POS B))
  (EPD (END T1 B) (END T2 B))       )

(VOLTAGE (PAR LN : T1 T2 DVLT RVLT)
  (ICS (END T1 LN) (END T2 LN) (VOLTCON T1 DIR DVLT)
    (VOLTCON T2 RES RVLT))
  (EID (VOLTCON T2 RES RVLT))
  (EIA (VOLTCON T2 DIR DVLT))
  (CCS (VOLTCON T1 DIR DVLT) (END T1 LN) (END T2 LN))
  (EPD (VOLTCON T2 DIR DVLT))
  (EPA (VOLTCON T2 RES RVLT))       )
```

FIG. 13. WorldE scenarios.

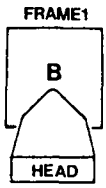
The switch being thrown is described by scenario MOVE. When the user wishes the switch to be thrown, a MOVE relation is added to the SWM. A MOVE relation consists of MOVE followed by the name of the switch to be thrown and the requested position. If the switch is not already in the requested position it will be moved to it.

Once the switch is thrown into the up or down position the scenario SWITCH asserts the new connection. Connections such as this are described in the model by stating that the two connected terminals are ends of the same line. The switch blade forms a new line when it is thrown. Therefore, SWITCH asserts that the switch blade is a line between the switch terminals.

When the new line is created by throwing the switch, the voltage change is handled by the scenario VOLTAGE. When one terminal of a line is directly connected to some power source and the terminal at the other end of the line is connected through a resistance, the latter terminal is changed so that it is connected directly with the same voltage as the former terminal.

4.3. THE TURING WORLD

This final example is a *Turing Machine* (Korfhage, 1966). The particular machine simulated is one of the five 3-state machines for the *Busy Beaver* problem. The Busy Beaver problem is to determine the maximum (finite) number of X's that can be written on a tape given n states (not including the halt state) and two symbols (B and X).




```
(STATE Q0)                    (TUPLE Q0 B X R Q1)
(HEAD FRAME1)                 (TUPLE Q0 X X R Q0)
(ON-TAPE FRAME1 B)            (TUPLE Q1 B X L Q2)
(EXTREMITYR FRAME1)           (TUPLE Q1 X X H HALT)
(EXTREMITYL FRAME1)           (TUPLE Q2 B X R Q0)
                              (TUPLE Q2 X X L Q1)
```

FIG. 14. Initial WorldT and SWM.

The initial SWM (Fig. 14) consists of rules describing the machine, a read-write head positioned over one frame of blank (denoted B) tape, and the state in which the machine is to start. Each rule contains five parts; the state the machine must be in for the rule to apply; the character that must be on the frame of tape positioned under the read–write head; the character to be written; the direction (including H for halt) in which the head is to move; and the next state the machine is to assume.

When the simulator is started, the TURING scenario (Fig. 15) looks for a tuple which applies to the current state and character. When one is found the designated character is written on the tape, the STATE relation is removed, and a MOVE relation is added to the SWM designating the direction in which the head is to move and the next state to be assumed.

If a MOVE relation exists which calls for the machine to halt, scenario MOVEH (note that MOVEH has no parameters) asserts that the machine is in the HALT state. If a MOVE relation exists calling for the head to be moved to the right (left), scenario MOVER (MOVEL) applies. It moves the head to the right (left) if a frame of tape exists there. It then waits for one unit of time before asserting the next state. This forces time to act as a counter of state changes.

Scenario R-3M (L-3M) manufactures a new frame of tape when a request has been made to move the head right (left) and no frame currently exists there. Generation of a new frame is made possible by modifying the use of the E-variable assignment in the

ICN. Instead of calculating a numeric value for EFRAME, a unique symbolic name is generated (by the LISP function GENSYM) and assigned to EFRAME. EFRAME is then asserted as a blank frame, connected to the right (left) of the previous frame, and the new right (left) extremity of the tape.

```
(TURING (PAR LABEL READ HPOS : WRITE DIR NEXT)
  (ICS (HEAD HPOS) (ON-TAPE HPOS READ) (STATE LABEL)
    (TUPLE LABEL READ WRITE DIR NEXT))
  (EID (ON-TAPE HPOS READ) (STATE LABEL))
  (EIA (ON-TAPE HPOS WRITE) (MOVE DIR NEXT))       )

(MOVEH
  (ICS (MOVE H *))
  (EID (MOVE H *))
  (EIA (STATE HALT))       )

(MOVER (PAR HPOS : NEXT TOPOS)
  (ICS (MOVE R NEXT) (HEAD HPOS) (CON HPOS TOPOS))
  (EID (HEAD HPOS) (MOVE R NEXT))
  (EIA (HEAD TOPOS))
  (CCN FUNC (PLUS # 1))
  (EPA (STATE NEXT))       )

(MOVEL (PAR HPOS : NEXT TOPOS)
  (ICS (MOVE L NEXT) (HEAD HPOS) (CON TOPOS HPOS))
  (EID (HEAD HPOS) (MOVE L NEXT))
  (EIA (HEAD TOPOS))
  (CCN FUNC (PLUS # 1))
  (EPA (STATE NEXT))       )

(R-3M (PAR HPOS : EFRAME)
  (ICS (MOVE R *) (HEAD HPOS) (EXTREMITYR HPOS))
  (ICN (:= EFRAME (GENSYM)))
  (EID (EXTREMITYR HPOS))
  (EIA (ON-TAPE EFRAME B) (CON HPOS EFRAME) (EXTREMITYR EFRAME)) )

(L-3M (PAR HPOS : EFRAME)
  (ICS (MOVE L *) (HEAD HPOS) (EXTREMITYL HPOS))
  (ICN (:= EFRAME (GENSYM)))
  (EID (EXTREMITYL HPOS))
  (EIA (ON-TAPE EFRAME B) (CON EFRAME HPOS) (EXTREMITYL EFRAME)) )
```

FIG. 15. WorldT scenarios.

This Turing Machine will halt when state Q1 is achieved and X is on the current frame of the tape. The final result is a tape six frames long filled with X's (Fig. 16).

Other Turing Machines can be simulated by starting with a different set of rules. For this purpose a request to assume an undefined state is also a halting condition.

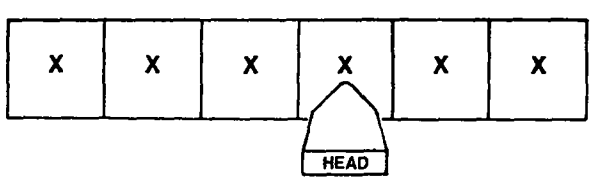


FIG. 16. Final State of WorldT.

5. The implementation

As in the implementation described in Hendrix (1973), neither string variables nor an equation solver were implemented. Hendrix (1973) suggests a technique for avoiding the use of string variables. The mechanisms available for equation solving were already discussed in the section on modeling. Additional incomplete areas are discussed later.

An elementary knowledge of LISP is assumed (McCarthy, 1960). The uninitiated should see Friedman (1974) or Siklóssy (1976).

(name (PAR pp_1 pp_2..... : sp_1 sp_2 .....)
(ICS (ics_1) (ics_2) .....)
(ICN (icn_1) (icn_2) ....($icne_1$) ($icne_2$) ....)
(EID (eid_1) (eid_2) .....)
(EIA (eia_1) (eia_2) .....)
(EGS (egs_1) (egs_2) .....)
(EGN (egn_1) (egn_2).....)
(CCS (ccs_1) (ccs_2) .....)
(CCN (ccn_1) (ccn_2).....)
(EPD (epd_1) (epd_2) .....)
(EPA (epa_1) (epa_2) .....))

Note: *icne* is an *icn* that does E-variable assignment.

FIG. 17(a). Form of input scenario.

NAME: name

PAR: ((pp_1 pp_2 ...) (sp_1 sp_2 ...))

IC: (((ics_1) (ics_2) ...)
(((icn_1) (icn_2) ...) (($icne_1$) ($icne_2$))))

PREFFECTOR: ((EID (eid_1) (eid_2) ...)
(EIA (eia_1) (eia_2) ...)
(EGS CBC (egs_1) (egs_2) ...)
(EGN CBC (egn_1) (egn_2) ...)
(CCS CBC (ccs_1) (ccs_2 ...))

C2N: (Boolean (ccn_1) (ccn_2) ...)

POSTEFFECTOR: ((EPD (epd_1) (epd_2) ...)
(EPA (epa_1) (epa_2) ...)
(DEL–CCS CBD (ccs_1) (ccs_2) ...))

FIG. 17(b). Form of internal system scenario.

5.1. SCENARIO CREATION

When the user inputs a scenario to the system it is used as an argument to CR–SC. CR–SC converts the user form of the scenario (Fig. 17(a)) into the internal system form and adds it to the global scenario list, GS–LIST. Internal scenario forms (Fig. 17(b)) are data objects of the data type named SCENARIO with fields NAME, PAR, IC, PREFFECTOR, C2N and POSTEFFECTOR. Each of these fields is filled according to the scenario supplied by the user.

The NAME field is filled with the CAR of the input scenario. The input scenario is then searched for the list beginning with the atom PAR. When it is found two lists are formed. One list is made of all those parameters preceding the colon. The other list is made of the remaining parameters. These lists are the primary parameter list and the secondary parameter list, respectively. If there were no secondary parameters the secondary parameter list is NIL. The PAR field is filled with these lists.

The IC field is filled by searching the input scenario for the lists beginning with the atoms ICS and ICN. Those relations following ICS form the symbolic initiation conditions list. Those found after ICN are separated into two groups, those relations which have := as their first element and those that do not. Those containing := are the ICN relations which assign E-variables their values. The numeric initiation conditions list is made of the normal ICN group along with the E-variable assignment group. Finally the IC field is filled with the ICS and ICN lists.

The PREFFECTOR is made up of a number of the input scenario fields. These consist of EID, EIA, EGS, EGN and CCS. Each of these atoms is defined as a function. When a SCENARIO's initiation conditions are satisfied, EVAL is mapped down the PREFFECTOR. The PREFFECTOR is formed by retrieving each scenario part mentioned above. If any part is not found, the field name is listed without any arguments. The functions are defined so that if they are called without any arguments, they will evaluate to some non-nil value. The functions EGS, EGN and CCS require an additional argument. This argument is CBC. Before these functions are evaluated, the control block being created is bound to CBC.

The C2N field contains information about the two types of continuation conditions that can be present. This information is used to determine the interrupt time of the control block being created from this scenario. The first element is a T or NIL; T meaning that the scenario contained no CCS and NIL meaning that it did. The remainder of the C2N field is made of relations found after the atom CCN in the input scenario. If there are no CCN in the input scenario, C2N will only be the listed Boolean value. If C2N is a LISTed T, the interrupt time will be the same as the creation time since the absence of continuation conditions signifies an instantaneous process. If C2N is a LISTed NIL, the interrupt time will be infinity, since it is a continuous process that will continue unconditionally until one of the CCS is no longer satisfied. Otherwise there are some CCN and the interrupt time is determined from them.

Finally there is the POSTEFFECTOR. It is much like the PREFFECTOR except that it is executed by mapping EVAL down it when the associated control block is being destroyed. Its first part is the CCS retrieved from the input scenario preceded by the atoms DEL–CCS and CBD. DEL–CCS is a function that undoes what was done by CCS. Before it is evaluated the control block being destroyed is bound to CBD. The remaining part of the POSTEFFECTOR is composed of the EPD and EPA of the input scenario. The functions DEL–CCS, EPD and EPA are defined so that they return non-nil values when given no arguments just as the functions in the PREFFECTOR are.

5.2. SWM CREATION

The SWM is stored on the property lists of the atoms EXPRS and SKLRS (Fig. 18). Those relations which contain no Y-variables are stored under EXPRS and those that do under SKLRS. The first atom of each relation is the indicator under which the remainder

of the relation is stored. If one or more relations have the same first atom, these relations less that first atom are stored in a list under that first atom. The function ADDEXP does this for those relations to be stored under EXPRS. ADDSKL does it for those stored under SKLRS.

A list is actually placed at the end of each relation before it is stored. For SKLRS the list contains the control block which has the equations describing how the value of the Y-variables change with time. For EXPRS it is a list of control blocks that are being *sustained* by the relation. A relation sustains a control block when that relation is a member of the CCS of the scenario associated with that control block. If the relation is ever deleted, each process described by the control blocks in the sustains list are interrupted.

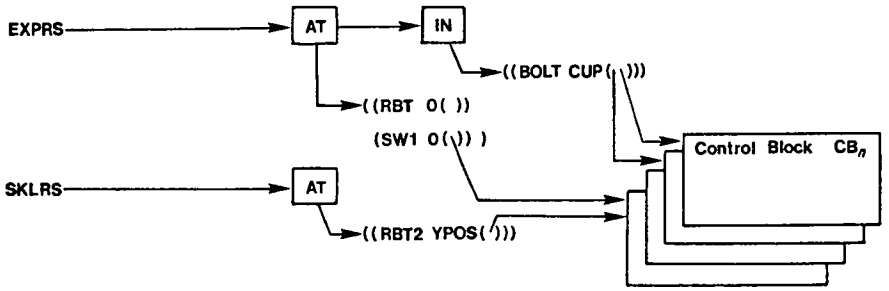


FIG. 18. SWM storage.

There are a few additional functions used to manipulate the SWM. DELEXP along with DES–EXP remove relations from EXPRS. DELSKL does the same for skeleton relations. CADDEXP only adds an explicit relation to the SWM if one like it does not already exist. CADDSKL results in a fatal error if the skeleton relation to be added is like one already existing. If there is an explicit relation like the skeleton relation, the explicit relation will be deleted before the skeleton relation is added.

5.3. CONTROL BLOCK CREATION AND DESTRUCTION

Control blocks are created by the function CR-CB. CR–CB takes a scenario and a list of bindings for the parameters as its arguments. A CONTROL–BLOCK (Fig. 19) is a data type defined with fields SC, BNDS, PCENT, INT, SKL–SUS and YEQS. CR–CB creates a CONTROL–BLOCK and adds it to the global control block list, GCB–LIST. It stores the scenario in the SC field and the binding list in the BNDS field. The current model time, GTIME, is stored under PCENT. PCENT is the creation time denoted in scenarios by %. The PREFFECTOR of the scenario is then executed after the bindings have been substituted for the parameters and the CONTROL–BLOCK for CBC.

As discussed earlier, PREFFECTOR execution is accomplished by mapping EVAL down it. EID and EIA make the appropriate additions and deletions to the SWM when executed. EGS adds each of the skeleton relations to the SWM by calling CADDSKL and adds all of those skeletons to the SKL–SUS field of the CONTROL–BLOCK. This list of skeletons is maintained so that their deletion from the SWM when the process is interrupted will be easier. EGN adds each of the Y-variable equations to the YEQS

field of the CONTROL–BLOCK. CCS adds the CONTROL–BLOCK to the sustains list of each explicit relation following it. If any of the relations cannot be found in the EXPRS of the SWM, the INT field of the CONTROL–BLOCK is set to GTIME. INT is where the interrupt time of the CONTROL–BLOCK is stored.

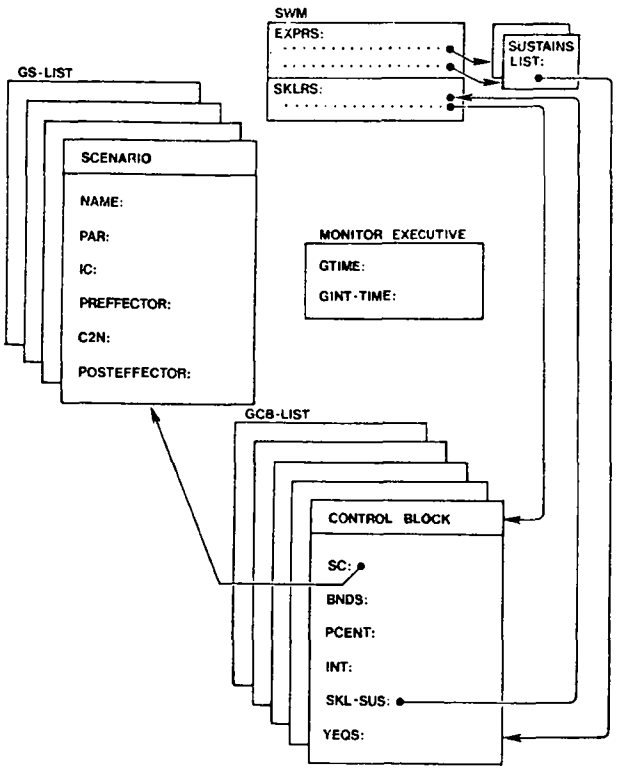


FIG. 19. Hendrix simulator structure.

If the INT field has not yet been set, CR–CB sets it to the result of calling DET–INT with the binding list, C2N of the scenario and YEQS of the CONTROL–BLOCK. DET–INT checks for the cases described earlier under C2N of scenario creation. If none of these simple cases hold, it checks to see if FUNC appears in the C2N portion. If it does, the interrupt time is determined by substituting the bindings for the parameters in the remainder of C2N and evaluating it. If FUNC does not appear, CR–CB forms the simultaneous equations necessary to determine the interrupt time from the C2N and YEQS and then asks the user for the solution.

Control blocks are destroyed by the function DES–CB. DES–CB calls the function DISMANTLEQS with the SKL-SUS of the CONTROL–BLOCK. It in turn calls SK–TO–EX. SK–TO–EX determines the values of the Y-variables at the current time. These values are then substituted back in the relation for the Y-variables. This results in an explicit relation. DISMANTLEQS then deletes these skeleton relations and adds the explicit relations just formed to the EXPRS section of the SWM.

After DISMANTLEQS has been called, the POSTEFFECTOR is executed by mapping EVAL down it after all bindings have been substituted for their respective parameters and the CONTROL–BLOCK for CBD. DEL–CCS removes the CONTROL–BLOCK from all of the sustains lists of the relations which had it added by CCS. EPD and EPA delete and add all of the appropriate relations to the SWM. DES–CB concludes by removing the CONTROL–BLOCK from GCB–LIST.

5.4. MONITOR EXECUTIVE

After the SWM and the scenarios have been input to the system by ADDEXP and CR–SC, control is passed to the monitor executive. The monitor executive is realized as a function of no arguments called MONITOREX (Fig. 20). In order to present MONITOREX it will first be necessary to discuss the role of the auxiliary functions to MONITOREX.

```
(DE MONITOREX ( )
  (REPEAT (SETQ CBD (FIND-INTCB GCB-LIST))
          (COND
          ((REPEAT WHILE (SETQ TSB (PRESCENARIO))
                         (OR (CH-TIME (CAR TSB)) (RETURN (MONITOREX)))
                         (CR-CB (CDR TSB))
                         (SETQ CBD (FIND-INTCB GCB-LIST))
                   UNTIL (EQ-TIME-INT) )
           (REPEAT       (DES-CB CBD) (SETQ CBD (FIND-INTCB GCB-LIST))
                   WHILE (EQ-TIME-INT) )      )
          ((INF-INT) (OR (MORE) (RETURN (ENDNOTE))))
          ((CH-TIME GINT-TIME)
           (REPEAT       (DES-CB CBD) (SETQ CBD (FIND-INTCB GCB-LIST))
                   WHILE (EQ-TIME-INT) )      )      )))
```

FIG. 20. MONITOREX function.

The auxiliary function PRESCENARIO determines if there is a scenario and binding set which represents a process that can be initiated before the earliest interrupt time of the control blocks. If there is, PRESCENARIO returns the initiation time, the scenario and the bindings for the process which can be initiated the earliest; otherwise it returns NIL.

The function FIND-INTCB returns a control block with the earliest interrupt time. It does this by scanning the GCB–LIST and comparing the INT fields. Besides returning the control block with the earliest interrupt time, FIND–INTCB also sets the global interrupt time, GINT–TIME, to that interrupt time. If there are no control blocks, GINTIME is set to INFINITY.

Two predicate auxiliary functions are EQ–TIME–INT and INF–INT. They both test the time at which the next interruption of a process is to take place, GINT–TIME. EQ–TIME–INT is true if the current model time is equal to that interrupt time and INF–INT is true if that interrupt time is INFINITY.

The remaining auxiliary functions are CH–TIME, MORE and ENDNOTE. CHTIME changes model time to the value of its argument. MORE passes control to the command level. If the user gives the GO command, MORE returns T. If the user gives the STOP command, MORE returns NIL. Otherwise MORE obeys the command and loops. ENDNOTE returns the final message concerning the time the modeling process was terminated.

MONITOREX is a large repeat loop. It first determines the control block with the earliest interrupt time and then evaluates a COND. The first COND pair is a REPEAT WHILE–UNTIL (Wise, Friedman, Shapiro & Wand, 1975), followed by a REPEAT UNTIL. The first of the pair, while PRESCENARIO succeeds, changes model time to the time returned by PRESCENARIO, creates a control block from the scenario and bindings returned by PRESCENARIO, and finds the control block with the earliest interrupt time. Then if the interrupt time of that control block is equal to the current model time, the second part of the COND is evaluated. The second part of the pair destroys the control blocks with the earliest interrupt while their interrupt is equal to the current time.

When PRESCENARIO fails the next COND pair is evaluated. If the earliest interrupt time is INFINITY, MORE is called. If MORE returns NIL, the user wishes to stop simulation so ENDNOTE is returned as the value of MONITOREX. If MORE does not return NIL, the main loop is entered again.

If the interrupt time was not INFINITY, model time is set to it. Control blocks with that interrupt time are then interrupted. When they have been interrupted the main loop is again entered.

CH–TIME is actually a little more complex than previously disclosed. It checks that a break point or snapshot time is not going to be passed by advancing time to the new value. If the new time will not pass these, GTIME is simply set to that new time and CH-TIME returns a non–nil value. If a snapshot time is about to be passed, GTIME is first advanced to the snapshot time, the picture is given, and GTIME is set to the new time and CH–TIME returns a non–nil value. If a break point is about to be passed, GTIME is moved up to the break point, MORE is called to allow the user to operate at command level, and NIL is returned as the value of CH–TIME so that the main loop will be re-entered.

5.5. PRESCENARIO

As stated earlier, before the monitor executive interrupts any given process, it first determines if there is a process which can be initiated before the global interrupt time. If there is, the process with the earliest initiation time is initiated. The monitor executive then recalculates GINT–TIME and again determines if any process can begin before that new time. This search for a scenario that can be initiated before the interrupt time is handled by the function PRESCENARIO. PRESCENARIO which involves almost half of the code is the heart of the entire system. The user who understands how PRESCENARIO works has a better understanding of how scenarios should be written.

PRESCENARIO returns NIL if no process can be initiated before GINT-TIME, otherwise it returns the scenario, its set of bindings, and the creation time for that process. PRESCENARIO (Fig. 21) carries this out by means of the major auxiliary functions EXPRESC, SKPRESC, BINDER and EBINDER.

BINDER is the first to be executed. It attempts to find bindings for the non-E-variable parameters of each scenario in GS–LIST which satisfy the scenario's symbolic initiation conditions. If no such bindings can be found for any scenario, BINDER returns NIL. Otherwise BINDER returns a list of pairs where each is a scenario and a successful binding set.

BINDER accomplishes this task by calling BINDERS with each scenario in GS-LIST. Since BINDERS returns all of the binding sets that satisfy the ICS of the given scenario, BINDER pairs the scenario with each of its binding sets before calling BINDERS with the next scenario. These pairs make up the list it returns.

After BINDERS has confirmed that those ICS which contain no variables are satisfied, it builds a list of binding sets. BINDERS builds this list in the variable BND. To begin with, the bindings for the first variable that satisfy the ICS of the scenario are found. Each of these bindings is paired with the parameter. These pairs are then stored in BND as the beginnings of possible binding sets. For example, if p_1 is the first parameter and bindings (b_{11} b_{12} b_{13}) are found, then BND is set to ($((p_1\ b_{11}))$ $((p_1\ b_{12}))$ $((p_1\ b_{13}))$).

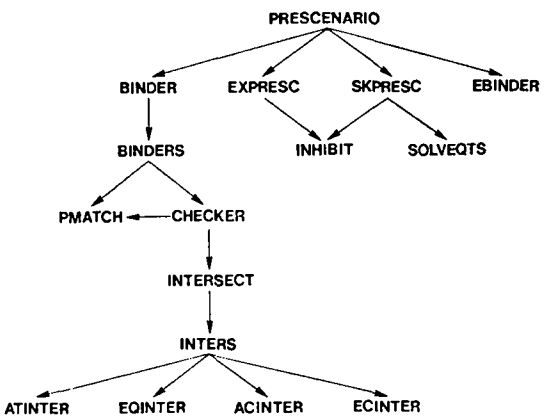


FIG. 21. Function map.

Now for each partial set of bindings in BND, BINDERS attempts to find a binding for the next parameter that is consistent with the previously bound parameters in that binding set. A binding is consistent with previously bound parameters if the ICS are still satisfied when all of the bindings are substituted for their respective parameters. If such a binding is found, it is added to the binding set. If more than one binding is found, the binding set is copied as many times as is necessary in order to have a set to which each new binding can be added. If no binding can be found for the new variable, that binding set is deleted from BND. So if two bindings b_{21}, b'_{21}, were found consistent with $((p_1\ b_{11}))$, one binding, b_{22}, with $((p_1\ b_{22}))$, and no bindings with $((p_1\ b_{13}))$, BND would be changed to ($((p_1\ b_{11})\ (p_2\ b_{21}))$ $((p_1\ b_{11})\ (p_2\ b'_{21}))$ $((p_1\ b_{12})\ (p_2\ b_{22}))$).

When this has been completed for all variables, BND contains the list of all binding sets that satisfy the ICS of the given scenario. This list is then returned as the value of BINDERS. If no binding set remains, its value will be NIL.

It remains to be explained how BINDERS finds the bindings for each new variable. This is accomplished by first forming a list of all symbolic initiation conditions that contain the new variable. Then the bindings for the previously bound variables, which are found in the given binding set, are substituted for their associated variables in this list. The symbol ** is substituted for the variable that is to be bound and * is substituted for all other variables.

Now the first element of this list is sent to PMATCH. PMATCH uses this as a pattern where * and ** match anything. PMATCH finds all matches for this pattern in the SWM and returns a list of those things which matched with **; thus returning a list of possible bindings for the variable of interest.

The list of possible bindings for the variable along with the rest of the symbolic initiation conditions list containing all substitutions is sent to CHECKER. CHECKER calls PMATCH with each successive element of the symbolic initiation conditions list. After each call to PMATCH, the intersection between the original list passed and the one just generated is taken. Elements of the original list that are not included in the intersection are removed from the list. This continues until a list of bindings is found that satisfies all of the ICS. This list of bindings is then returned.

PMATCH determines that a relation from the SWM matches the pattern if each element of the pattern matches each element of the relation. Each element is tested by the function M1. M1 returns NIL if the two elements do not match and returns T if they do match. In addition, if there are any conditions to the match they are stored in the variable CNDS.

There are a number of ways that two elements can match. If both are identical LISP atoms they match. If the pattern element is * or ** it is an automatic match. Here if ** matches a Y-variable it is the Y-variable's associated equation that is actually matched. If the pattern element is a number it also matches a Y-variable but with a condition. The condition is the resulting equation for # formed by replacing the Y-variable with the number in the equations. This newly formed equation is placed in CNDS signifying that the Y-variable only matches the number at time #. If the pattern element is itself an equation (resulting from the binding of a previous variable to a Y-variable's equation) and the relation element is a number, it matches again with the condition formed by replacing the Y-variable in the equation with the number. Lastly, if the pattern element is an equation and the relation element is a Y-variable, they match with the condition that some value for # can be found such that the values of the Y-variable and the Y-variable from the equation are EQUAL.

It is now apparent that PMATCH does not always return a list of simple bindings for the variable. Each binding in the list may be an atom, an equation, or an atom or equation listed with a *time-equation* from CNDS. For this reason the intersection in CHECKER is not a simple intersection but one that must take equations and conditions into account. This intersection operation is done by the function INTERSECT. INTERSECT uses the functions INTERS to determine if two elements match. INTERS first checks to see if both elements are atoms. If they are, they must be the same atom in order to match. If only one of the elements is an atom the function ATINTER is called. If neither element is an atom INTERS checks to see if one of the elements is an equation. If so EQINTER is called. If neither element is an atom or equation but one is a paired atom and condition, ACINTER is called. Otherwise both elements must be paired equations with conditions, in which case ECINTER is called.

ATINTER, EQINTER, ACINTER and ECINTER all determine if the two elements they are passed match. They must take into account that some elements may only match by making further restrictions. For example, a number and an equation containing a Y-variable and # match, but they match only at some value for # which satisfies the equation after the number is substituted for the Y-variable. So the number and equation match only at those times which satisfy the resulting time-equation. That is, they match

a third structure that is the number paired with the condition represented by the time-equation. Because two elements can match but only with regard to a third element, these functions return the element which both match instead of just T. If they do not match, NIL is returned.

There are other elements that match besides two identical atoms and a number and equation. One equation matches another but with a condition. The value of the Y-variable in the equations must be equal. A time-equation relating this is constructed and returned along with one of the equations. All other cases are formed by having one or both of the elements paired with a time-equation. These time-equations are simply collected along with any that are produced by number and equation matchups and paired with the matching element when returned. For example, if one element is a number with a time-equation and the other is an equation paired with a time-equation, the number and equation result in the number and a time-equation as previously discussed. The other two time-equations along with this newly produced one are grouped and returned along with the number.

It is BINDERS' responsibility to later purify the binding list of those time-equations. BINDERS collects all time-equations, removing them from their paired atom or equation, and places them at the end of the binding list as if they were bindings for the atom *EQTS*. These time-equations will later be used to determine the time that the associated scenario can be used to initiate a process.

This completes the explanation of BINDERS. In review, BINDER calls BINDERS with each scenario in GS–LIST and forms a list of scenarios paired with binding sets that satisfy the ICS of the associated scenario. These bindings are found by means of the functions PMATCH and CHECKER. PMATCH makes an initial list of bindings according to one initiation condition and CHECKER eliminates all of those bindings that are inconsistent with the other ICS.

The list of scenarios and bindings is sent to the function EXPRESC. EXPRESC looks for a scenario binding set pair that has no equations as bindings and no *EQTS* binding. When such a scenario binding set pair is found, EXPRESC determines if the bindings also satisfy the ICN. It does this by substituting the bindings for the associated parameters in the ICN and then evaluates each. If they all are true, the ICN are satisfied. If anyone of them is NIL, the ICN are not satisfied and that scenario binding set pair is removed. When all numeric conditions are satisfied, EXPRESC next checks to see if the process described by the scenario and binding set is already underway. The function INHIBIT accomplishes this by looking through the GCB–LIST for a control block utilizing the same scenario and primary parameter bindings. If such a control block is found, the scenario binding set pair is removed. Otherwise EXPRESC has found a process that can be initiated at the current model time. EBINDER is then called to add the E-variable bindings to this binding set. Since the current time is necessarily the earliest time at which a process can be initiated, this scenario and binding set along with GTIME are immediately returned as the value of PRESCENARIO.

When EXPRESC is unsuccessful in finding a scenario binding set pair as described above, it passes the list of remaining possibilities to SKPRESC. This means that the list passed to SKPRESC contains binding sets with *EQTS* bindings, equations as bindings, or both. SKPRESC currently eliminates all of those which have equations as bindings (to be explained). SKPRESC takes the remaining list of scenario binding set pairs and determines which satisfy both the associated scenario's ICN and INHIBIT.

For those that do, SKPRESC uses the function SOLVEQTS to solve the time-equations bound to *EQTS* for the earliest model time equal to or greater than GTIME. The scenario binding set pair with the earliest solution time is returned by PRESCENARIO after the E-variables have been bound by EBINDER as the process which can be initiated at the earliest time.

5.6. COMMAND LEVEL

MORE is the function that interprets each command it is given. If it is given a command that it does not recognize, it informs the user that an error has been made and asks the user for the command again.

The GO and STOP commands are the only ones that do a return from MORE. All others result in the main loop of MORE being reentered. The GO command returns T and the STOP command returns NIL.

TRACE and UNTRACE set the global variable TRACE. TRACE is set to the scenario names that are being traced. The functions CR–CB and DES–CB are sensitive to the value of TRACE. When the scenario name is a member of TRACE they print out the relevant trace information.

SNAPSHOT sets the global variable GPT to the ordered list of model times making sure that all those times are yet to come. When CH-TIME is called it makes sure that the time it is to change GTIME to is less than the first time in GPT. If it is not CH-TIME advances GTIME to that first time in GPT, takes a picture, and calls CH-TIME after having removed the first time from GPT.

AUTOSNAP and UNSNAP set the global variable AUTO to T and NIL, respectively. If AUTO is true, CH–TIME determines if the new time it is about to change GTIME to is in GPT. If it is not in GPT the time is added to GPT and CH–TIME is called. If it is in GPT, CH–TIME takes a picture as usual. If AUTO is NIL, this check is not made and GPT is not affected.

ACTION maps the function ACTION down the GCB–LIST. This prints the name of the scenario and the binding list associated with each control block.

SCENARIOS prints the NAME field of each SCENARIO on the GS–LIST. TIME prints the value of GTIME.

PICTURE calls the function PICTURE. It prints the value of GTIME, the relations stored under EXPRS without the sustains lists, and the relations under SKLRS without the control block references and with the current Y-variable values substituted.

BREAK checks the time given to make sure that it is greater than GTIME. If it is not, the command is ignored. If it is greater, the global variable BREAKP is set to that time. When CH–TIME is called it checks to see if model time is going to be advanced past the time in BREAKP. If it is, GTIME is set to the time in BREAKP and MORE is called. When MORE returns, CH–TIME returns NIL so that the main loop of MONITOREX will be re-entered.

ADD and DEL are realized as calls to EIA and EID, respectively. The relations following ADD and DEL are used as the arguments of EIA and EID.

EPSILON affects the value of the global variable EPSILON. EPSILON is set to whatever value it is given.

6. Deviations in the implementation from the general model

6.1. ICN DIFFERENCE

The implementation of the ICN is not as general as that described by Hendrix (1973) in his conceptualization of the system. In his version the ICN and ICS are both used simultaneously in order to find a binding set which satisfies both. Either the ICN or the ICS could force a binding set to be altered.

In our implementation, the ICN are only used as a restriction on the binding sets obtained from satisfying the ICS. After the possible binding sets which satisfy the ICS have been determined, each is checked with the ICN. Those which do not satisfy the ICN are discarded. The remaining binding sets are those used by the system.

```
(SETALARM (PAR R KLOCK CSTIME : LOC)
  (ICS (ALLOCATE-ACTIVATE R SETALARM KLOCK CSTIME) (AT R LOC)
    (AT KLOCK LOC))
  (ICN (LT # CSTIME) (GT 12 (DIF CSTIME #)))
  (EID (ALLOCATE-ACTIVATE R SETALARM KLOCK CSTIME) (ALARM OFF KLOCK))
  (EIA (ALARM SET KLOCK))      )

(SOUNDALARM (PAR KLOCK CSTIME)
  (ICS (ALARM SET KLOCK CSTIME))
  (ICN (EQUAL CSTIME #))
  (EID (ALARM SET KLOCK CSTIME))
  (EIA (ALARM SOUNDING KLOCK))      )

(OFFALARM1 (PAR R KLOCK : LOC)
  (ICS (ALLOCATE-ACTIVATE R OFFALARM KLOCK) (AT R LOC)
    (ALARM SOUNDING KLOCK) (AT KLOCK LOC))
  (EID (ALLOCATE-ACTIVATE R OFFALARM KLOCK) (ALARM SOUNDING KLOCK))
  (EIA (ALARM OFF KLOCK))      )

(OFFALARM2 (PAR R KLOCK : LOC CSTIME)
  (ICS (ALLOCATE-ACTIVATE R OFFALARM KLOCK) (AT R LOC)
       (ALARM SET KLOCK CSTIME) (AT KLOCK LOC))
  (EID (ALLOCATE-ACTIVATE R OFFALARM KLOCK)
       (ALARM SET KLOCK CSTIME))
  (EIA (ALARM OFF KLOCK))      )
```

FIG. 22. Additional WorldH scenarios.

This means that the ICN are incapable of altering any of the bindings contained in a binding set. This forces the user to write some scenarios in a different way. In the previously presented world model that was used by Hendrix (1974) there was a portion deleted (Fig. 22). This portion describes an alarm clock which the robot can turn on and off and set for times less than 12 units of time away.

The ICN of SOUNDALARM does not work as described because the ICN of this scenario needs to alter the binding set and initiation time except if the binding is being attempted at the model time when the alarm is to sound. This follows from the fact that # is always bound to the current model time. If the current time is less than the time the alarm is set for and there is another process which has an initiation time greater than the time the alarm is set for, the alarm will never sound. The ICN of SOUNDALARM will fail since the current model time is not the time at which the alarm is to sound. So the system will assume that the other process is the one which can be initiated the earliest. Model time will be moved to that process's initiation time and that process will begin, skipping the time at which the alarm is to sound.

There is a simple remedy to this problem. The remedy follows from the fact that SOUNDALARM works properly if bindings are attempted at the time the alarm is to sound. The remedy is in the form of an additional scenario (Fig. 23) which guarantees that the monitor executive will stop and look for a process to initiate at that time.

```
(MONITORALARM (PAR KLOCK CSTIME)
  (ICS (ALARM SET KLOCK CSTIME))
  (ICN (NOT (EQUAL CSTIME #)))
  (CCS (ALARM SET KLOCK CSTIME))
  (CCN FUNC CSTIME)      )
```

FIG. 23. Scenario MONITORALARM.

This new scenario starts when all conditions are right for the original scenario except the time. The process this scenario describes has an interrupt time equal to the time the original scenario can initiate its process. Therefore this scenario guarantees that the modeling process is stopped at the appropriate time.

The ICN (GT 12 (DIF CSTIME #)) of SETALARM is similar to the one in SOUNDALARM. An additional scenario (Fig. 24) can be written for it in the same way.

```
(MONITORSET (PAR R KLOCK CSTIME : LOC)
  (ICS (ALLOCATE-ACTIVATE R SETALARM KLOCK CSTIME) (AT R LOC)
    (AT KLOCK LOC))
  (ICN (LT # CSTIME) (LE 12 (DIF CSTIME #)))
  (CCS (ALLOCATE-ACTIVATE R SETALARM KLOCK CSTIME) (AT R LOC)
    (AT KLOCK LOC))
  (CCN FUNC (DIF CSTIME (DIF 12 (TIMES 1.1 EPSILON))))
```

FIG. 24. Scenario MONITORSET.

6.2. EQUATIONS AS BINDINGS FOR SCENARIO PARAMETERS

We stated earlier that binding sets which contain equations as bindings are currently discarded by the system. Such a binding signifies that the value that the parameter is bound to varies with time. Such bindings are potentially useful, their exclusion is only evidence that more work needs to be done in order to fill this gap in the implementation.

Equations appearing as bindings can be handled in at least two different fashions. One is to pick a particular time, solve the equations with that time, and use the resulting values as the bindings. The time would be chosen by calculating the earliest time at which the values of the equations fulfil the ICN of the scenario. This time would also be the initiation time of the process. Such a situation might result from a scenario that describes how one robot hands something to another robot and its use when the robots are moving.

Another possibility is that the equation bindings signify that one process is dependent on another. Hendrix (1974) has considered this situation even though it does not appear in his article. For this he created another type of variable, X-variables. X-variables represent dynamic entities being defined by equations in other processes. Except for their definition in the scenario, they are used just as Y-variables. They may appear in the CCN portion of a scenario. His example of a scenario using X-variables is LOC (Fig. 25). LOC describes the location of an object being grasped by a moving robot. The location

of the grasped object is always the same as the robot's. When the control block describing the robot's movement is destroyed, so must the control block describing the location of the object.

```
(LOC (PAR OBJ : RBT XLOC)
  (ICS (GRASPING RBT OBJ) (AT RBT XLOC))
  (EGS (AT OBJ YLOC))
  (EGN (:= YLOC XLOC))     )
```

FIG. 25. Scenario LOC.

7. Finer points of modeling

7.1. MODEL CONSISTENCY

When creating a model world, care must be taken that the model is consistent. A model world is consistent if for any two processes the interruption of one of the processes does not keep the other process from being initiated at that same time. For a model world that means that if there are two processes which have their initiation conditions fulfilled at the same model time, the adds and deletes caused by the initiation of one of the processes must not make the initiation conditions of the other process unsatisfied.

WorldH is an inconsistent model. The inconsistency concerns the scenarios MOVABILITY and GOTO. If a robot has just grasped an immovable object and has been given an ALLOCATE-ACTIVATE order to GOTO some point, two different things can happen. If the MOVABILITY process is initiated first, the GOTO process will not be initiated. If the GOTO process is initiated first, the MOVABILITY process will be initiated but after the robot's movement has already begun. Therefore the model will have the robot moving while it is immovable. Since theoretically the monitor executive's choice of which process to initiate first is arbitrary, the SWM resulting from this situation is not predictable. This is why the model is said to be logically inconsistent.

Since the implementation of the monitor executive does not make an arbitrary choice as to which process to initiate next, some inconsistent model worlds can be made to operate as if they were consistent. The monitor executive always initiates the process whose associated scenario was most recently input. So the inconsistent situation in the previous example can be avoided by inputting the GOTO scenario before the MOVABILITY scenario. The monitor executive will then always initiate the MOVABILITY process before the GOTO process, making the resulting SWM predictable.

7.2. ICS EFFICIENCY

Two logically equivalent model worlds may operate at different levels of efficiency because of the ordering of the ICS in the scenarios. This is a result of the way in which BINDERS operates. For each parameter in the scenario it extracts all the ICS that contain it. It then attempts to find each of those relations in the SWM in the same order that they were extracted. If any of those relations are not found, the remaining ones are not tried since that binding set could not possibly satisfy all the ICS of the scenario. For this reason, the monitor executive will operate faster if those relations in the ICS which have a greater chance of failing are placed before the other relations in the ICS of the scenarios. Because of this it is usually best to place the ALLOCATE-ACTIVATE order as the first relation in the ICS.

7.3. CONCLUSION

Since Hendrix's paper appeared, there have been a number of significant developments in other works which directly bear on some of his ideas. Here, we briefly mention some of these without attempting a complete review.

The work by Scragg (1975), developed independently from that of Hendrix, contains a number of similar ideas relating to process characterization. Rieger (1976) characterizes processes through a set of primitive event types and a set of primitive links. Although Sacerdoti (1975) treats time as a discrete phenomenon, his work represents the state of the art for those type systems. Babich, Grason & Parnas (1975) present an event-driven simulation system in the context of automobile movement. Finally, Young (1976) compares the Hendrix monitor executive approach to a distributed monitoring approach where each entity "understands" its own state and controls its own related processes. For example, in the distributed model, the bucket itself determines when it begins to overflow. In the Hendrix system, the monitor executive makes such determinations.

This paper presented a self-contained introduction and implementation description to a simulation system for modeling simultaneous actions and continuous processes (Hendrix, 1973). An operating environment for the system and its implementation are discussed. The power of the system is increased by allowing general LISP expressions to be included in selected portions of scenarios. The generality of the system (including Turing power) is demonstrated through its application to a number of sample robotic and non-robotic worlds. Model consistency and an additional variable type are described. The binding method for scenarios and other major implementation issues are discussed. The LISP code for the entire system and the output resulting from additional sample runs are available in an earlier version of this paper (Lowrance & Friedman, 1975).

We appreciate the critical reading of Ben Shneiderman, Stuart C. Shapiro, Jonathan V. Post and Elliot M. Soloway. Stuart C. Shapiro gave us a mini-review course in physics which helped clarify the electrical circuit example. The success of this project owes much to personal communications with Gary G. Hendrix.

References

BABICH, A. F., GRASON, J. & PARNAS, D. L. (1975). Significant event simulation. *Communications of the Association for Computing Machinery*, **18**, 323–330.

FIKES, R. E. & NILSSON, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem-solving. *Artificial Intelligence*, **2**, 189–208.

FRIEDMAN, D. P. (1973). GROPE: A graph processing language and its formal definition. *Ph.D. Thesis.* Department of Computer Science, The University of Texas at Austin.

FRIEDMAN, D. P. (1974). *The Little LISPer.* Palo Alto: Science Research Associates.

GAINES, B. R. (1975). 3.65 Simulation of natural systems. *Computing Reviews*, 159–160.

HENDRIX, G. G. (1973). Modeling simultaneous actions and continuous processes. *Artificial Intelligence*, **4**, 145–180.

HENDRIX, G. G. (1974). Personal communication.

KORFHAGE, R. R. (1966). *Logic and Algorithms.* New York: John Wiley and Sons, Inc., pp. 106–114.

LOWRANCE, J. D. & FRIEDMAN, D. P. (1975). The Hendrix model of simultaneous actions and continuous processes: an introduction and implementation description. *Technical Report Number 33.* Department of Computer Science, Indiana University.

MCCARTHY, J. (1960). Recursive functions of symbolic expressions and their computation by machine—I. *Communications of the Association for Computing Machinery*, **3**, 184–195.

QUAM, L. H. (1969). *Stanford LISP 1.6 Manual.* Stanford Artificial Intelligence Project, Stanford University.

RIEGER, C. (1976). On oganization of knowledge for problem-solving and language comprehension. *Artificial Intelligence*, **7**, 89–127.

SACERDOTI, E. D. (1975). A structure for plans and behavior. *Technical Note 109.* Artificial Intelligence Center, Menlo Park, California: Stanford Research Institute. (To be published by American Elsevier.)

SCRAGG, G. W. (1975). Answering questions about processes. In NORMAN, D. A. & RUMELHART, D. E., *Explorations in Cognition.* San Francisco: W. H. Freeman and Company, pp. 349–375.

SIKLÓSSY, L. & DREUSSI, J. (1973). An efficient robot planner which generates its own procedures. *3rd International Joint Conference on Artificial Intelligence*, Stanford, pp. 423–430.

SIKLÓSSY, L. & ROACH, J. (1973). Proving the impossible is impossible is possible: Disproofs based on hereditary partitions. *3rd International Joint Conference on Artificial Intelligence*, pp. 383–388.

SIKLÓSSY, L. (1976). *Let's Talk LISP.* Englewood Cliffs, New Jersey: Prentice Hall, Inc.

WINOGRAD, T. (1972). *Understanding Natural Language.* New York: Academic Press.

WISE, D. S., FRIEDMAN, D. P., SHAPIRO, S. C. & WAND, M. (1975). Boolean-valued loops *BIT*, **15**, 431–451.

YOUNG, R. M. (1976). Design choices for world-modeling systems. *Proc. AISB Summer Conference on Artificial Intelligence and Simulation of Behavior.* University of Edinburgh Press, pp. 376–386.